DD CEN/TS 15121-1:2011

BSI Standards Publication

Postal Services — Hybrid Mail

Part 1: Secured electronic postal services (SePS) interface specification — Concepts, schemas and operations

raising standards worldwide™

National foreword

This Draft for Development is the UK implementation of CEN/TS 15121-1:2011.

This publication is not to be regarded as a British Standard.

It is being issued in the Draft for Development series of publications and is of a provisional nature. It should be applied on this provisional basis, so that information and experience of its practical application can be obtained.

Comments arising from the use of this Draft for Development are requested so that UK experience can be reported to the international organization responsible for its conversion to an international standard. A review of this publication will be initiated not later than 3 years after its publication by the international organization so that a decision can be taken on its status. Notification of the start of the review period will be made in an announcement in the appropriate issue of *Update Standards.*

According to the replies received by the end of the review period, the responsible BSI Committee will decide whether to support the conversion into an international Standard, to extend the life of the Technical Specification or to withdraw it. Comments should be sent to the Secretary of the responsible BSI Technical Committee at British Standards House, 389 Chiswick High Road, London W4 4AL.

The UK participation in its preparation was entrusted to Technical Committee SVS/4, Postal services.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© BSI 2011

ISBN 978 0 580 70585 4

ICS 03.240

Compliance with a British Standard cannot confer immunity from legal obligations.

This Draft for Development was published under the authority of the Standards Policy and Strategy Committee on 31 January 2011.

Amendments issued since publication

Date	Text affected

TECHNICAL SPECIFICATION

SPÉCIFICATION TECHNIQUE

TECHNISCHE SPEZIFIKATION

CEN/TS 15121-1

January 2011

ICS 03.240

English Version

Postal Services - Hybrid Mail - Part 1: Secured electronic postal services (SePS) interface specification - Concepts, schemas and operations

Postalische Dienstleistungen - Hybride Sendungen - Part 1: Schnittstellen-Spezifikation für Gesicherte elektronische Postdienste (SePS) - Begriffe, Schemata und Betrieb

This Technical Specification (CEN/TS) was approved by CEN on 9 August 2010 for provisional application.

The period of validity of this CEN/TS is limited initially to three years. After two years the members of CEN will be requested to submit their comments, particularly on the question whether the CEN/TS can be converted into a European Standard.

CEN members are required to announce the existence of this CEN/TS in the same way as for an EN and to make the CEN/TS available promptly at national level in an appropriate form. It is permissible to keep conflicting national standards in force (in parallel to the CEN/TS) until the final decision about the possible conversion of the CEN/TS into an EN is reached.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: Avenue Marnix 17, B-1000 Brussels

Contents

Page

Foreword

This document (CEN/TS 15121-1:2011) has been prepared by Technical Committee CEN/TC 331 “Postal Services”, the secretariat of which is held by NEN.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

According to the Memorandum of Understanding (MoU) between the UPU and CEN, signed Oct. 22nd, 2001; 3.3 CEN notifies the following deviation from the source text:

> The term *"postal administration"* meaning a postal service designated by one member country of the UPU was changed according with the wording of the Postal Directive to *"postal service"*.

This document is the equivalent to Part 1 of a multi-part UPU standard, S43: Secured electronic postal services (SePS) interface specification. S43 was originally published as a single part standard covering only one secured electronic postal service, but has been split into parts to allow the standard to be extended to cover other services based on the same concepts, schemas and operations. Part 1 defines these concepts, schemas and operations.

Part 2 defines EPCM Services, and uses the specification of Part 1.

The specification is complemented by five annexes. Annex A and Annex B are normative; Annex C, Annex D and Annex E are informative. The specification contains a Bibliography.

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to announce this Technical Specification: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

Introduction

This interface specification describes a standardized way for postal services or its system development teams to build a secured electronic postal services (SePS) capability which can be offered to customers as part of an electronic service inventory.

A SePS is a postal service which is accessed electronically through the use of an interface based on an appropriate subset of the operations (verbs) specified in this document. Together these define a set of standardized application layer software security services aimed at facilitating the introduction and integration of the following capabilities into a 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 SePS events;
- logging of user events deemed relevant to the business transaction;
- tying together of SePS events into a business transaction Lifecycle;
- retrieval of evidence data in support of dispute resolution and future challenges in a non-repudiation context.

Individual SePS services may support different subsets of the defined operations. For example, the electronic postal certification mark (EPCM) service, defined in part B of the standard (see UPU standard S43b) uses the CheckIntegrity, PostMark, RetrieveResults, Sign and Verify operations to support the capture and reproduction of evidence data attesting to the fact that a business transaction was conducted and completed in an environment of integrity and trustworthiness.

The process of integrating SePS features into an automated application is termed "SePS-enabling" the target application. Each call to a SePS can be looked at as a non-repudiable SePS event or SePS transaction within the application's overall business workflow. These non-repudiable 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.

This specification describes the SePS interface standard and contains four main clauses and five annexes:

Clause No	**Description of content**
5	*Key SePS concepts:* introduces a number of key concepts which are drawn on in the remainder of the specification;
6	*Overview of SePS operations*: provides an overview of the standard operations, supported by the schema defined in Annex A, which can be combined to implement secured electronic postal services which comply with this specification;
7	*Common schema types used across SePS operations*: defines common WSDL element types that are sent to and returned from the SePS;
8	*Detailed specification of SePS operations*: provides a detailed definition of the operations which were introduced in Clause 6;
Annex A	*(normative) SePS XML Schema V1.15:* provides the formal XML Schema for the SePS interface;
Annex B	*(normative) Web Service Description Language (WSDL) V1.15:* provides the formal WSDL specification of the SePS interface;
Annex C	*Examples:* provides specific examples illustrating the various constructs used within the interface;
Annex D	*(informative) European and international standards inter-relationships and evolution:* provides background information on other signature standards which exist in the same domain as the SePS interface specification. Their influence and role in shaping this standard and its evolution is also covered;
Annex E	*(informative) Relevant intellectual property rights (IPR):* provides information about intellectual property rights whose use has been reported as possibly being implied by certain implementations of the specification.

The implementation of part or all of this specification might involve the use of intellectual property that is the subject of patent and/or trademark rights. It is the responsibility of users of the standard to conduct any necessary searches and to ensure that any pertinent rights are in the public domain; are licensed[1)] or are avoided. Neither CEN nor the UPU can accept any responsibility in case of infringement, on the part of users of this document, of any third party intellectual property rights. Nevertheless, document users and owners of such rights are encouraged to advise the Secretariat of the UPU Standards Board and/or of CEN/TC 331 of any explicit claim that any technique or solution described herein is protected by such rights in any CEN or UPU member country. Any such claims will, without prejudice, be documented in the next update of this standard, or otherwise at the discretion of the Standards Board, respectively CEN/TC 331. Annex E of this document lists the intellectual property rights brought to the attention of CEN/TC 331 and the UPU Standards Board prior to approval of the publication of this version of the standard.

NOTE The mention of intellectual property rights, in Annex E, is on a ‘without prejudice’ basis. That is, such mention indicates only that some party has expressed the view that use of the standard might, in some circumstances, infringe the mentioned intellectual property rights. It should not be taken as in any way confirming the validity of such view and users should conduct their own searches to determine whether the mentioned IPR is in fact applicable to their specific case.

1) Mail service contractors are advised to ensure that reliance on intellectual property that is not in the public domain does not inadvertently lead to the creation of an effective monopoly. This could occur, even if usage of the intellectual property concerned is licensed by the mail service contractor, unless the terms of the licensing agreement commit the IPR holder to making licences available, on appropriate terms, to the mail service contractor's customers and suppliers, including competitors of the IPR holder.

1 Scope

This document specifies a standard XML interface that will enable software applications to call a secured electronic postal service (SePS), provided by a postal service, which is based on the concepts, schemas and operations described herein.

The specification provides:

— a definition of standard operations which can be combined to support secured electronic postal services;

— a full description of all mandatory and optional request parameters required for use of these operations;

— a full description of all response elements and the detailed circumstances under which they are returned.

The specification also 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) specification. The versions of these applicable at the date of publication of this version of the specification are contained in this document as Annex A and Annex B respectively. These can also be obtained in electronic format from the UPU Technical Standards CD-ROM or from the UPU Standards Secretariat.

In case of any conflict between Annex A and other provisions of this specification, Annex A shall be regarded as definitive.

The SePS schema specification in Annex A is discreet and version specific. Postal Services are free to select which discrete interface versions they support. However, except in the case of upgrades to V1.15 adopted to ensure cross-border compatibility, postal services who upgrade from older versions of the schema (e.g. from V1.14) to a newer one are required to support backward compatibility of previously supported versions of the SePS interface specification as it applies to both processing requests/responses and honoring previously issued PostMarkedReceipts. Individual posts are free to address this backward compatibility challenge as they see fit.

The `Version` element which is present in every request and which is included in the `PostMarkedReceipt` can be used to support this backwards compatibility requirement.

The requirement for backward compatibility does not apply to cross-border scenarios where V1.15 has been adopted to ensure compatibility. The SePS Interface specification includes a digital signature platform supporting basic cryptographic service operations as well as a comprehensive framework for the delivery of evidentiary, witnessing, and non-repudiation services. The specification provides for 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.

SePS 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 specification:

— complies with IETF RFC 3161 in respect of time stamp tokens, time stamp values and other time stamp attributes;

— complies with all mandatory requirements (i.e. qualified as "required" or "shall" in the text) of IETF RFC 3126 and ETSI TS 101 733 as they apply to Electronic Signatures – Complete (i.e. ES-C);

— complies with the IETF RFC 2630 ASN.1 layout for all PKCS objects utilised in the specification;

— supports XMLDSIG signature formatting as defined in IETF RFC 3275;

— complies with IETF RFC 2560 in respect of the `ValidationData` element.

This version of the specification does **not** cover:

— a description of the issues surrounding inter-operability between multiple postal SePS implementations when a business transaction Lifecycle requires the participation of more than one SePS implementation in a cross-border scenario involving two or more postal services;

— issues surrounding SePS 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, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 UPU standards

UPU Standards glossary

NOTE 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.

2.2 Internet Engineering Task Force (IETF) documents

NOTE 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, U.S.A. Tel: (+1 703) 620 8990, Fax: (+1 703) 620 9071, www.ietf.org. RFCs can also be obtained from www.faqs.org/rfcs/.

RFC 2315[2] – PKCS #7 Version 1.5

RFC 2560[3] – X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP (June 1999), M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams

RFC 2617 – HTTP Authentication: Basic and Digest Access Authentication (June 1999) J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart

RFC 2822 – Internet Message Format

2) This defines the ASN.1 layout for all relevant PKCS objects utilised by the SePS.

3) The ValidationData element defined in the WSDL interface specification (Annex B) refers directly to this RFC. If a SePS 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.

RFC 3126[4] – Electronic Signature Formats for long term electronic signatures (September 2001), D. Pinkas, J. Ross, N. Pope

RFC 3161[5] – Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP) (August 2001), C. Adams, P. Cain, D. Pinkas, R. Zuccherato

RFC 3275[6] – XML-Signature Syntax and Processing (March 2002), D. Eastlake, J. Reagle, D. Solo

RFC 3447 – Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1

2.3 Organization for the Advancement of Structured Information Standards (OASIS)

NOTE 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. OASIS specifications can be obtained via its web site, www.oasis-open.org, which also provides contact details for written communications.

oasis-sstc-saml-core-1.1 – Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1. OASIS Standard, September 2003, 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 (DSS) Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dss

OASIS DSS – Digital Signature Services (DSS) - EPM Profile, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dss

3 Terms and definitions

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

3.1
asymmetric cryptographic algorithm
cryptographic algorithm using two related keys, a public key and a private key, and in which one key can decrypt what has been encrypted with the other one

3.2
certificate (e.g. X509)
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

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 SePS interface specification honours ES-C mandatory requirements, i.e. qualified as "required" or "shall" in the text.

5) References within the SePS WSDL specifications (Annex B) 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.

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 WSDL interface specification (Annex B) allows for both PKCS7 binary ASN.1 formatting of signature objects as well as the more recent XMLDSIG signature formatting.

3.3
certificate Revocation List (CRL)
list of revoked certificates

3.4
certification
process of creating a public key certificate binding an entity's identity to its public key

3.5
certification Authority (CA)
entity trusted by one or more other entities to create, assign, revoke or suspend public key certificates

3.6
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

3.7
cross-certification
mechanism by which two CAs exchange certificates to implement a trusted relationship

3.8
cryptographic key
parameter controlling the operation of a cryptographic function

3.9
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

3.10
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

3.11
distinguished name
globally unique name for an entity

3.12
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.

3.13
entity
CA, RA or end entity

3.14
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 used as input to a digital signature algorithm.

3.15
key
see cryptographic key

3.16
key pair
set of keys, consisting of a public key and a private key, that are associated with an entity in a public key cryptography system

3.17
non-repudiation
service providing proof, beyond reasonable doubt, of the integrity and origin of data which can be validated by a third party

3.18
object identifier
sequence of integer components identifying an object such as an algorithm or attribute type

3.19
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

3.20
Registration Authority (RA)
entity responsible for the identification and authentication of certificate subjects but which does not sign or issue certificates

3.21
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

3.22
RSA algorithm
cryptographic method created by Rivest, Shamir, and Adelman for which the intellectual property rights are held by RSA Data Security

3.23
Secure Socket Layer (SSL)
protocol developed by Netscape for encrypted transmission over TCP/IP networks

3.24
subject
entity whose public key is certified in a public key certificate

3.25
subscriber
end entity or subject that is considered to have an account, with a SePS Provider, providing it with the ability to subscribe to the SePS as per some pre-determined contractual arrangement

3.26
Trusted Time Stamp (TTS)
record mathematically linking a data item to a date7 assured by a trusted time stamping authority

3.27
Time Stamp Authority (TSA)
trusted third party which issues and/or verifies trusted time stamps

3.28
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 and abbreviations

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)
EPCM	Electronic Postal Certification Mark

NOTE 1 The acronym EPCM refers to a particular Regulation adopted by the Postal Operations Council within the UPU and describes a specific SePS which complies with the specification in Part B of this standard (S43b). Other (future) Regulations and standards might specify other services which comply with this part of the standard (S43a).

IETF	Internet Engineering Task Force
LDAP	Lightweight Directory Access Protocol
NA	Not Applicable
OCSP	Online Certificate Status Protocol
PS	Postal Service
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, inventors of the RSA encryption algorithm owned by RSA Data Security, Inc.; frequently used to refer to the algorithm itself
SAML	Security Assertion Markup Language
SePS	Secured electronic postal service(s)

NOTE 2 The acronym SePS is used both to refer to a single implementation — a secured electronic postal service — and generically, to refer to services that comply with this specification. The appropriate interpretation (service or services) has to be determined from the context in which the acronym is used.

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
WSDL	Web Services Description Language
XAdES	XML Advanced Electronic Signatures
XSD	XML Schema Definition
XSLT	XML Stylesheet Language Transformation

5 Key SePS concepts

5.1 Authentication

The act of physically authenticating individual calls to a SePS is outside the scope of this specification. The SePS delegates all authentication mechanisms to an implementation-specific authentication facility which would normally sit out in front of the SePS. Each implementing post is free to select and implement its own authentication mechanism. It is, however, required that some level of basic or default authentication (see also `AccessLevel` = default and AccessLevel = `Signed`) be supported. Possibilities include, but are not limited to:

— the Basic Authentication as specified in the HTTP protocol and supported by all Web servers;

— a strongly-authenticated 2-way Mutual Authentication supported by certificates.

All sessions with the SePS 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, authentication would take place in the Web Server front-end, shielding the SePS Application Server from the need to perform authentication. If HTTP Basic Authentication is used as the `default` authentication mechanism, the SePS implementation (server implementation) 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 SePS 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 SePS 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 (and optionally the password) from the Authorization line and initialize the `BasicAuth` element before it travels on to the SePS application server (see element descriptions below). In either case, the SePS 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 SePS implementation can be freely determined by the implementing post.

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, based on the implementor’s and customers' preferences. Once authenticated, the ***logged-in*** user shall be passed back to the SePS implementation. This could be standardized and made independent of the authentication mechanism.

At the discretion of the implementor, the `AlternateIdentity` element can be a SAML Assertion, a Liberty Alliance Authentication Context, an X509 Certificate or any other identity token deemed suitable to the postal service concerned. This element should be initialized by the authentication service which executes in front of the SePS application server. Federated Identity Management servers could work in this fashion as well. This is consistent with the ***delegated authentication*** approach used by the SePS, 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 SePS 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 checking reasons). The Password has only been included for potential future use in unforeseen authentication scenarios and should not be required by the SePS Application Server since authentication has already taken place in the Web front-end. As such it is marked as nillable. See also ClaimedIdentity in 7.3.

5.2 Digital signature verification

Digital signatures are used both to verify the identity of the party submitting or registering an electronic document and to ensure the integrity of the document content. All input is maintained as evidence and can be re-verified at any point in the future should authenticity be challenged. Digital signature integrity and certificate status are verified using PKI-based digital fingerprinting and signature verification technologies to check for both content and certificate integrity.

5.3 Error handling

Error handling within the SePS 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 specified in TransactionStatus and TransactionStatusDetailType in 7.17.

5.4 Event logging

Event logging covers the storage of the evidence (i.e. escrow) data associated with business transactions. Service records are maintained by the service provider for as long as is needed to satisfy customer, postal service and local legal requirements.

EXAMPLE 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.

5.5 Lifecycle management

Because business transactions often involve multiple parties dealing with multiple documents over an extended time frame and often across country borders, the SePS is designed to be able to 'tie together' any number of events that are deemed 'of significance' to the participating parties. It does so by using the notion of business transaction Lifecycles, with every operation being considered to belong to a Lifecycle of (at least) one event. A Lifecycle can be started explicitly using the StartLifecycle operation (see 6.12), or can be started implicitly on the any operation by supplying a `TransactionKey` value previously returned by the SePS. In this way transaction events can be tied together into a Lifecycle. The latter is termed implicit use of Lifecycle.

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 7.10 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. 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 operation in order to get a `TransactionKey` back from the SePS and then use this `TransactionKey` on subsequent events. Thus events can be added to this Lifecycle by supplying the returned `TransactionKey` on subsequent calls. By default every operation is part of a Lifecycle of one event.

NOTE In order to extend a Lifecycle beyond a single event, the ExtendLifecycle flag has to be set in addition to specifying the `TransactionKey` of the Lifecycle to which this event should be added.

Access to the transactions in a Lifecycle can be limited to restricted groups or individuals through use of the `ParticipatingParty` complex element (see ParticipatingPartyType in 7.10 for details). If this is required, the Lifecycle has to be started explicitly, using StartLifeccycle (6.12); implicit Lifecycles can be used only when the user is willing to allow a value of 'Global' for the `AccessScope` element.

5.6 Non-repudiation

The non-repudiation service retains all customer-required tracking and evidence records of significance within the business transaction life cycle. These records are used to support the following non-repudiation services:

— 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 non-repudiation services ensure an extremely trustworthy end-to-end business transaction process. It is intended that the non-repudiation service, through the implementation of jurisdiction-specific legislative requirements, can act as a legally binding transaction notarization service both within and across postal domains.

Where required to do so, the service provider can provide any authorised individual or organization with any and all required evidence of the existence, integrity, and logged date 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.

5.7 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 to 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 `PostMarkedReceipt` 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 operation and return a `PostMarkedReceipt`. If `IssuePostMarkedReceipt` is specified in the RequestType element for such an operation, the `PostMarkedReceipt` will be returned in the operation's response.

5.8 Processing directives or options

Every SePS has an Options structure where the caller can specify the special handling directives to be performed for the 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`.

5.9 Protection of confidentiality

PKI-based encryption services are used to 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.

5.10 Time stamping

Signature verification services are time stamped with a unique `PostMarkedReceipt` or value attesting to the fact that the post providing the SePS stands behind the evidence gathered during the signing ceremony, as well as the subsequent verification status. Additionally the date at which the transaction was conducted is captured both in the logging facility (see 5.4) and within the verified signatures themselves.

5.11 Transaction handling

Every request/response within the SePS 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 SePS. Usually there is one per postal service, although operational considerations and/or jurisdictional issues could result in multiple instances.

The `Key` is the unique Lifecycle identifier within the domain of the service-instance concerned and is generated by the SePS when a new Lifecycle is created.

The Sequence identifies the particular request/response within the Lifecycle identified by the `Key`.

NOTE Transactions within the SePS are logical transactions. Each atomic operation is completely logged to the database as part of a single RDBMS logical unit of work. When multiple events need to be tied together, they are linked to form a Lifecycle. In practice, even an individual transaction is regarded as being part of a Lifecycle consisting of a single event. This is implicit, however, and the caller need do nothing special if multi-event Lifecycles are not required. Further details are provided in 5.5.

6 Overview of SePS operations

6.1 General

This clause provides an overview of the basic operations, or verbs as they are sometimes referred to, that can be combined to implement a SePS which complies with this specification. Each sub-clause describes one such operation; the corresponding sub-clause of clause 8 provides a detailed specification. A specific service will generally support only the subset of these operations which is required to provide the functionality which is relevant to the definition of the service concerned.

EXAMPLE Part B of this standard defines the EPCM service. This requires support for five core operations (CheckIntegrity, PostMark, RetrieveResults, Sign and Verify), with support for other operations being considered as extended service options. Further parts of the standard will, when developed, define other standardised service offerings, including a registered electronic mail service.

Table 1 — SePD operations, showing the applicable options for each operation

Operation / Option	CheckIntegrity	Decrypt	Encrypt	Locate	Log Event	Post Mark	RetrievePostalAttributes	RetrieveResults	RetrieveSummary	Sign	StartLifecycle	Verify
DecryptincomingEnvelope	✔		✔		✔	✔				✔		✔
EncryptResponse		✔				✔		✔		✔		✔
EndLifecycle		✔	✔	✔	✔	✔			✔	✔		✔
ExtendLifecycle		✔	✔	✔	✔	✔			✔	✔		✔
IssuePostMarkedReceipt	✔		✔					✔		✔		✔
ReturnSignatureinfo		✔	✔					✔		✔		✔
ReturnTimeStampAudit								✔				
ReturnX509Info		✔	✔	✔				✔		✔		✔
StoreNonRepudiationEvidence	✔	✔	✔			✔		✔		✔		✔
VerifyCertificate			✔	✔						✔		✔

Key: ✔Valid Option for the Operation

6.2 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 SePS'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 SePS 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` specifies the valid values for the type of eContent being compared. See the description of OriginalContentType in 7.9.

If senders require Proof of Delivery, implementers should use the CheckIntegrity operation rather than 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 supplies the document on the CheckIntegrity request, as well as signing over that content (see ClaimedIdentity in 7.3), 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 provided `OriginalContent` with the content which already exists in the non-repudiation database under the requested `TransactionKey`.

6.3 Decrypt

This operation (together with the Encrypt operation described in 6.4) provides native SePS support for confidentiality. By default, the SePS utilizes its own private decryption key for all requests which have specified the `DecryptIncomingEnvelope` option. Support for the use of customer-specific private decryption keys can be pre-configured in cases in which the SePS is deployed within a subscribing organisation. Postal services are free to choose whether or not to offer this deployment model to their subscribing customers.

6.4 Encrypt

6.4.1 General

This operation (together with the Decrypt operation described in 6.3) provides native SePS support for confidentiality. The SePS 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 primarily intended for use by organizational subscribers wishing to encrypt content for the parties with whom they are dealing. It would not normally be used by clients interfacing with the service from their desktops; where encryption is needed by such clients, it would normally be performed by the client desktop application.

Encryption can use any valid public key associated with the intended recipient.

6.4.2 Delegated Confidentiality Service

The "delegated confidentiality" service corresponds to a particular usage of the SePS’s confidentiality capability. This service frees individuals from having to manage the public keys of intended recipients. A subscriber wishing to encrypt content for confidentiality reasons simply encrypts that content with a public key provided by the postal service. This single public key belongs to the post and is the only public key the customer needs to know. After having encrypted the content with this post-specific public key, the result is secure for transport and can be sent to the recipient.

The recipient will of course not have the private key required to decrypt the received envelope and has to ask the post’s local SePS to Decrypt it. It would not normally make sense for the SePS to simply Decrypt the content and pass it back to the recipient in the clear. Provided that the recipient turns on the EncryptResponse option in the request, the SePS therefore decrypts the data and then re-encrypts the result using the recipient’s public key. To provide the SePS with the its public key, the recipient is obliged to sign the request. See 7.7 EncryptResponse Option for more detailed coverage.

6.5 Locate

This SePS operation supports a user friendly interface to public certificate retrieval. It is useful when a client does not have the public key of a recipient for whom it wishes to encrypt content.

6.6 LogEvent

The LogEvent operation allows subscribers to log and ***system*** timestamp any content they believe is of significance to the non-repudiation Lifecycle. This facility is also useful when the SePS is used in conjunction with other system components which support other events in the non-repudiation Lifecycle which need to be logged. For specific use-case examples of use of the LogEvent operation, please refer to 8.6.1.

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

6.7 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`.

The PostMark 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 at or before a particular date. The caller can pass in several content formats to be PostMarked and the SePS will return a `PostMarkedReceipt` optionally containing an RFC 3161-compliant timestamptoken as well as summary information and a signed receipt.

6.8 RetrievePostalAttributes

The RetrievePostalAttributes operation is used to access a list of localization attributes that are specific to a local SePS provider. This operation is used to retrieve a list of `Locator`-specific attributes by category, where each content 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 `Locator`-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.

6.9 RetrieveResults

The RetrieveResults operation is normally reserved for challenge-time requests. However SePS 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.

EXAMPLE The PosteCS service from Canada Post and La Poste (France) provides an example of this type of document pick-up service.

See also ClaimedIdentity in 7.3. 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 the signature verification operation for which the caller wants the results. For a list of all events in a Lifecycle, refer to the RetrieveSummary operation in 8.10.

6.10 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 operation in the Lifecycle are returned as an unbound repeating structure. The user can then select a specific event in the Lifecycle from the returned list and access the details for that event using the RetrieveResults operation.

6.11 Sign

The Sign operation is a '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 assure 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 is subsequently passed to the SePS for verification and PostMarking using the Verify operation.

6.12 StartLifecycle

The SePS 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 SePS is designed to be able to 'tie together' any number of events that are deemed 'of significance' to the participating parties.

Every operation is part of a Lifecycle of (at least) one event. The Lifecycle to which an operation belongs is specified by means of the TransactionKey element. Where this corresponds to a pre-existing value, the SePS will tie the operation and all its content and results to the key specified. For the first (or only) operation of a Lifecycle, the TransactionKey can be specified as null, resulting in the implicit starting of a Lifecycle, or use can be made of an explicit StartLifecycle operation. Implicit Lifecycles have a value of 'Global' for the AccessScope element, so an explicit StartLifecycle is required when the subscriber wishes to specify a ParticipatingParty list. Please refer to ParticipatingPartyType in 7.10 for details.

6.13 Verify

The Verify operation performs cryptographic verification of a supplied digital signature. This 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.

The Verify operation is invoked by the originator of a 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 (see 6.2) CheckIntegrity operation.

It should be noted that the encrypted versions (b) and (d) above are required to contain a signature after decryption. The decryption is specified by simply turning on the `DecryptIncomingEnvelope` option. The Verify operation can optionally return a `PostMarkedReceipt` by turning on the `IssuePostMarkedReceipt` option.

The receipt on a Verify operation is the postal service'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 is retained by the SePS.

NOTE The `PostMarkedReceipt` carries different semantic meaning when used in conjunction with a PostMark operation (see 6.7) where no signature is actually being verified. This difference arises from the fact that, since no signature verification has taken place, the postal service is only able to attest to the existence of a particular datum at a given date.

7 Common schema types used across SePS operations

7.1 Introduction

The following object types represent common WSDL element types that are sent to and returned from the SePS. 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.

7.2 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 7.10 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 SePS. `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 SePS 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 7.10 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.

7.3 ClaimedIdentity

ClaimedIdentity is an optional input element within the SePS 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 signing party. Please refer to 8.2 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 (email addresses, Distinguished Names, etc). This type is borrowed from oasis-sstc-saml-core-1.1 (http://www.oasis-open.org/committees/download.php/3406/oasissstc-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. It needs to be extended from the base type as shown in the YourFavoriteIdentityTokenType example above.

See 7.3 of oasis-sstc-saml-core-1.1 (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 1 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 illustrates how stronger non-repudiability is supported. 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.

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 2 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.

7.4 ClientApplication

The `ClientApplicationType` is used in requests to indicate the desktop client and version from which the SePS operation originated. Example: "`Microsoft Office OPC Signature Provider`". This element (through its attribute) also identifies the scheme used to transform the document. That is, the processing of any inclusions and/or exclusions of document parts, as well as any optional transformations 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 digest calculation. It's use is optional, and when not specified, the entire document will be assumed.

NOTE This element may be used to trigger special processing on the SePS Server. An example of this exists with Office 2007 where Microsoft uses a non-standard XMLDSIG Transform which is part of their Open Packaging Convention (OPC). This non-standard Transform forces SePS servers to suppress Manifest verification on the server in order to obtain successful verification results for the conventional non-Manifest References. The caller is expected to initialize the NameAndVersion element to "`Microsoft Office OPC Signature Provider`". The version has been left out of the string in the event that Office 2009 for example maintains this transform.

```
<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>
```

7.5 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 might wish to filter transactions on.

7.6 ContentMetadata

This element can be optionally passed as input on most operations and can be used by 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... A similar element named ReceiptMetadata is 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>
```

NOTE This metadata content will not be saved in the SePS's non-repudiation database unless the caller sets the associated operation's StoreNonRepudiationEvidence option flag to true.

7.7 EncryptResponse Option

This option can be utilized in three different SePS operations: Verify, RetrieveResults and Decrypt.

EncryptResponse is a Special Case optional usage of the SePS confidentiality capability, and works in conjunction with a local encrypt operation issued by an individual client customer using the SePS's public encryption key. This is termed "Delegated Confidentiality" within the SePS context. Usage of delegated confidentiality frees individuals from having to manage the public keys of intended recipients. A subscriber wishing to encrypt content for confidentiality reasons simply encrypts that content locally with a public key provided by the postal service. This single public key belongs to the post and is the only public key the customer needs. Content which has been encrypted with this post-specific public key is secure for transport; the envelope containing it can be sent to the recipient.

The recipient does not have the private key required to decrypt the received envelope and needs to ask the post's SePS to Decrypt it. It would not make sense for the SePS to simply Decrypt the content and pass it back to the caller in the clear. The SePS therefore "encrypts the response" when the `EncryptResponse` option is turned on in the recipient's request. To do this, the SePS requires the callers public key. To provide this, the caller signs the request. This usage is termed "Sign for Pickup" which may also optionally employ the `EncryptResponse` option.

EncryptResponse is often utilized on a Verify request, on a RetrieveResults request, or on a Decrypt request. The recipient may turn on the `EncryptResponse` option on a RetrieveResults operation and "pickup" a document signed by an originator which was left with the SePS.

Another scenario involves a CheckIntegrity call, by the recipient, in which the content is passed to the SePS 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 described below, three scenarios involving end-to-end confidentiality need to be considered. They differ primarily in how the document is transported and which operation the recipient uses. Only the first scenario ensures irrefutable Proof-of-Delivery and Proof-of-Possession:

a) **Scenario:** "Sender encrypts document with SePS public key and delivers document directly to recipient". The sequence of events is as follows:

   1) Sender locally signs the document;

   2) Sender locally encrypts the signed document with the SePS's public key;

   3) Sender calls the SePS to Verify the document using the following options:

      i) IssuePostMarkedReceipt;

      ii) DecryptIncomingEnvelope;

      iii) EncryptResponse to protect the updated signature;

   4) Sender decrypts the Verify response using its own private key and checks status;

   5) Sender re-encrypts the signed document for the recipient using the recipient's public key (assumed to be either available or retrieved using the SePS's Locate operation);

   6) Sender mails the signed and encrypted document to recipient;

   7) Recipient decrypts the document with its own private key;

8) Recipient locally encrypts the signed and PostMarked document with the SePS's public key;

9) Recipient issues signed CheckIntegrity request with the following option:

   i) DecryptIncomingEnvelope;

10) Recipient checks status.

b) **Scenario:** "Sender encrypts document with SePS public key and deposits document with SePS for pickup". Sequence of events as follows:

   1) Sender locally signs the document;

   2) Sender locally encrypts the signed document with the SePS's public key;

   3) Sender calls SePS to Verify the document using the following options;

      i) IssuePostMarkedReceipt;

      ii) DecryptIncomingEnvelope;

      iii) EncryptResponse to protect the updated signature;

   4) Sender checks status;

   5) Sender notifies the recipient that the signed document can be picked up using the appropriateTransactionKey, which is passed to the recipient "out of band";

   6) Recipient issues signed RetrieveResults request with the EncryptResponse option;

   7) Recipient decrypts the response using its own private key to extract the document.

c) **Scenario:** "Sender Verifies document, re-encrypts verified document with recipient's public key, and delivers document directly to recipient". Sequence of events as follows:

   1) Sender locally signs the document;

   2) Sender locally encrypts the signed document with the SePS's public key;

   3) Sender calls the SePS to Verify the document using the following options;

      i) IssuePostMarkedReceipt;

      ii) DecryptIncomingEnvelope;

      iii) EncryptResponse;

   4) Sender decrypts the Verify response using its own private key and checks status;

   5) Sender re-encrypts the signed document for the recipient using the recipient's public key (assumed to be either available or retrieved using the SePS's Locate operation);

   6) Sender mails the signed and encrypted document to the recipient;

   7) Recipient decrypts the document with its own private key;

   8) Recipient locally verifies the signature and optionally the PostMark.

In situations in which the SePS 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 SePS implementations SHALL ensure that authenticated requests which involve decryption of sensitive content honor and respect restricted `ParticipatingParty` lists within that Lifecycle.

7.8 Event

This simple object type represents a list of all valid operations that the SePS will honor in support of the NotifyEvents element within the StartLifecycle operation (See Lifecycle management in 5.5 for more information.). This simple object is used by the participating parties (refer to ParticipatingPartyType in 7.10 for details) and is referenced through it's 'NotifyEvents' element. An event's value can be any ValidOperation.

7.9 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(s), excluding the bounding tags, shall be passed in on a single or successive `OriginalContent` element(s) 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>
```

7.10 ParticipatingPartyType

When working within a SePS lifecycle, the SePS 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 1 ClaimedIdentity used on its own can also provide POD/POP support.

Each `ParticipatingParty` would be accessing the SePS 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 1 CN=Joe Public,O=Acme,OU=Purchasing,C=CA.

Each `ParticipatingParty` identified by `PartyName` would be accessing the SePS 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 2 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 2 For `AccessScope = Organizational` and `AccessLevel = Default`, implementations would derive the authenticated user's organization from information captured at registration time, normally kept in the SePS's registration database. The design of this registration sub-system is beyond the scope of the SePS 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>
```

7.11 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. For example, when the `IssuePostMarkedReceipt` element is included on a Verify operation, the returned receipt is the Postal Service's attestation of having successfully performed the requested operation. In the case of a Verify, the receipt 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 Service'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 SePS's nonrepudiation capability. The `PostMarkedReceipt` contains three core pieces of information:

— the Receipt containing a timestamp (either represented as a TimeStampToken or as a TstInfo structure referenced within an XMLDSIG `PostMarkedReceipt` signature),

— the referenced user content being PostMarked, and

— a signature of authenticity binding everything together.

The XML layout of the `PostMarkedReceipt` reflects the signature type used to produce 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 covering the above 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.` XMLDSIG PostMarks will not be created over PKCS7 signatures and PKCS7 PostMarks wil not be created over XMLDSIG signatures. The XMLDSIG based `PostMarkedReceipt` is a formal `dsig:Signature,` however like other XML content, it is transported as an `epm:QualifiedDataType` of `MimeType text/xml.`

```
<xs:element name="PostMarkedReceipt">
    <xs:complexType>
        <xs:sequence>
            <xs:choice>
                <xs:element name="PKCS7SignedReceipt" type="epm:PKCS7SignedReceiptType"/>
                <xs:element name="XMLSignedReceipt" type="epm:QualifiedDataType"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<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="DigestAlgo" type="xs:string" nillable="true"/>
        <xs:element name="DataMimeType" type="xs:string"/>
        <xs:element name="PostMarkImage" nillable="true"/>
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="xs:base64Binary">
                        <xs:attribute name="Format" type="xs:string" default="JPG"/>
                        <xs:attribute name="Size" type="epm:ValidImageSize"default="Small"/>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <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 optional RFC 3161 timestamptoken taken over the appropriate content as dictated by the context in which the `PostMarkedReceipt` was requested. This element is mandatory for PKCS7-based `PostMarkedReceipts`. It is optional for XMLDSIG-based `PostMarkedReceipts` where the `TstInfo` Reference can serve this purpose and allows everything to be expressed as formal XMLDSIG signatures. Implementing posts are free to include the optional `TimeStampToken` in either case.

In the case of a `PostMarkedReceipt` requested on a `Verify` or a `Sign` operation via the `IssuePostMarkedReceipt` option or a direct PostMark operation of Types 4 or 5, the timestamp will be a “signature” timestamp taken over the SignatureValue 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 PostMarked. Please refer to the PostMark operation and the `MimeType` element for specific details.

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 (See standalone versus embedded `PostMarkedReceipts` below). 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 SePS is `sha1WithRSAEncryption` and **SHALL** be supported by SePS implementations when creating and verifying signatures of all types including the `TimeStampToken` described in this subclause.

NOTE 1 The value of the `MimeType` attribute of this element is `application/timestamp-token`. For the `MimeType` of the data that was PostMarked, please referto the `DataMimeType` element within the `Receipt`.

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 SePS. 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:

a) For `IssuePostMarkedReceipt` on a **Verify** operation of an XMLDSIG signature

   1) If the `Location` sub-element is specified as `standalone`, a standalone `PostMarkedReceipt` structure will be returned in the Verify response. See Example 1 – Standalone <PostMarkedReceipt> over a Verified Signature

   2) 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 together returned in the `SignatureData` element of the Verify response. See Example 3 – Embedded <PostMarkedReceipt> over a Verified Signature

b) For `IssuePostMarkedReceipt` on a **Verify** operation of a CMS/PKCS7 signature

   1) 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.

2) 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 together 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. The content and the `PostMarkedReceipt` will be returned together as part of the `PostMarkedReceipt` structure.

See Annex C for a description of layouts and references of both types. Please also refer to the `PostMark` operation in 8.5 for more details on supported content types.

ReceiptSignature – This optional element contains a PKCS7 detached signature over the contents of the Receipt. The `ReceiptSignature` is only used for receipts issued in response to PKCS7-based operations. It is a SePS 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 7.3 apply here as well.

NOTE 2 The default signature creation algorithm for the SePS is `sha1WithRSAEncryption` and **SHALL** be supported by SePS 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>Production</Environment>
</Locator>
<Key>041019-133230-59841915</Key>
<Sequence>1</Sequence>
</TransactionKey>
<Requester>.....</Requester>
<Operation>Verify</Operation>
<TimeStampValue>20040327174718Z</TimeStampValue>
<RevocationStatusQualifier>CRL Checked</RevocationStatusQualifier>
<TimeStampToken MimeType="application/pkcs7-signature">.....</TimeStampToken>
<MessageImprint>.....</MessageImprint>
<DigestAlgo>sha1</DigestAlgo>
<DataMimeType>text/plain</DataMimeType>
<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 SePS 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 C.

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. SePS implementations are obliged to initialize this element. Please also refer to the discussion dealing with the verification of the `PostMarkedReceipt` in 8.13.3 under the PostMarkedReceipt topic. It is not required for XMLDSIG-based PostMarkedReceipts.

DigestAlgo – This element specifies the hash algorithm used when creating the digest value of the content being PostMarked. For PostMarks of Type 1 and 2, this element specifies the hash algorithm used by the SePS when creating the `MessageImprint` field of the `TimeStampToken` (PKCS7) or the `PostMarkedContent` element (XMLDSIG). Possible values include: `sha1` or `md5`.

DataMimeType – This element is the `MimeType` of the incoming `Data` element saved from the original PostMark operation. It is required when `PostMarkedReceipts` are subsequently sent back in to a SePS to be verified. It will also be initialized by a SePS for `PostMarkedReceipts` issued as a result of the `IssuePostMarkedReceipt` option. Please refer to the specific operation for details.

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 SePS 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 SePS to create a `PostMarkImage` and return it in this response element.

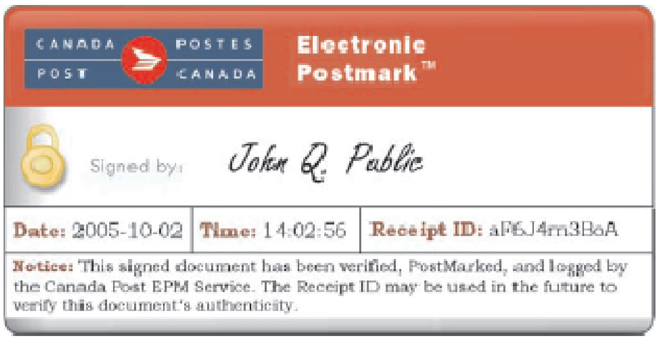
ReceiptMetadata – This optional unbound element provides a vehicle for Posts to add any country-specific textual or binary evidentiary content to be included in the `PostMarkedReceipt`. It is covered by the `ReceiptSignature` and therefore shall be prepared on the SePS server when the `PostMarkedReceipt` 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 SePS operations which support the `PostMarkedReceipt` i.e. Sign, Verify, RetrieveResults, CheckIntegrity, and Encrypt. The presence of this option instructs the SePS to generate a `PostMarkedReceipt` and include it in the response. This element as 3 sub-elements described below.

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

PostMarkImage – This optional boolean element when set to `true` instructs the SePS 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 3 This image is not to be confused with the Universal SePS Logo which is a non-PostMark-specific, non-Transactionspecific branding graphic for the SePS.



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.

Figure 1 — Example of a PostMark image

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 SePS to return one of the supported sizes available from this post's SePS implementation. Valid values are `Small`, `Medium`, and `Large`. Each size is specified by the Postal Service in pixel width and height and is consistent across Postal SePS implementations. Please consult your local Postal Service for details. This information can also be extracted from the Postal "Yellow Pages" directory using the RetrievePostalAttributes operation.

7.12 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 enveloping 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 C for further details.

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

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 DSS documents for further details.

A `TstInfo` element contained in an `Object` element referred to by the first `Reference` will be created by the SePS. 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. SePS 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 SePS.

Similarly a 2nd `Object` element exist for the `Receipt` structure which includes the `Receipt`, and optionally 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 optional for XMLDSIG-based receipts. The `TstInfo` and `Receipt` structures are described in Annexes C.2 and C.3.

3rd <Reference> element – is one whose URI attribute references the dsig:Object containing 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 requests a PostMark, the 3rd `Object` element will **contain** (if standalone) or **refer to** (if embedded) the `SignatureValue` element(s) of the signature(s) being PostMarked. The SePS will Verify all signatures specified in the `SignatureSelector` element. If the `SignatureSelector` element is omitted, the SePS 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.

When a `PostMarkReceipt` is embedded in the signed document being verified, this 3rd Reference element does not point to a copy of the user's SignatureValue(s) as is the case with a standalone `PostMarkedReceipt`, but rather points directly to the `SignatureValue's` themselves by means of a filtering Xpath expression. Copying the `SignatureValue's` is not required as the `PostMarkedReceipt` is within the signed document. For a detailed desription of this convetion, see Example 3 – Embedded <PostMarkedReceipt> over a Verified Signature.

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 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.

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 which allow the inclusion of the `PostMarkedReceipt` without invalidating the original signature can be explored by the client. This could be accomplished by explicit `Transform`'s which exclude appropriate nodesets.

7.13 QualifiedDataType

The QualifiedDataType is used throughout the SePS schema in both request and response elements. 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`, `application/timestamp-token`, etc… When used in responses, the `MimeType` attribute of `QualifiedDataType` will be initialized based on the actual content of the element being returned. For example, when the `QualifiedDataType` is used for a returned `PostMarkedReceipt` which is of type XMLDSIG, then the `MimeType` attribute will be set to `text/xml`. Similarly if the `QualifiedDataType` is used for a returned `TimeStampToken` then the `MimeType` will be set to `application/timestamp-token`.

```
<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>
```

7.14 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 on which the operation was performed. A SignatureInfoType complex element is populated and returned by the SePS. 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 selected elements in this structure are not used.

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 SePS implementation for informational purposes only and will not affect interoperability. Supported values are standardized but remain SePS implementation specific. The value `sha1WithRSAEncryption` is the default PKCS1 signature creation algorithm and **SHALL** be supported by SePS implementations when creating and verifying signatures. This signature algorithm is also the default for cross-border signatures sent between the SePS of other postal services. 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 include: `sha256WithRSAEncryption` or `sha512WithRSAEncryption`. Please refer to RFC 2315 – PKCS #7 Version 1.5 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 SePS implementation specific. SePS 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 date 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 SePS to Verify it, this element will contain the signing date as created by the client application. This value is provided in the standard UTC Format – YYYYMMDDHHMMSS followed by 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. It is 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>
```

7.15 SignaturePolicyIdentifier

This abstract type cannot be directly implemented. A specific case is extended from the 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.

Use of this abstract type might be required in an implementation for which 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>
```

7.16 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 5.5 for more information. Within the domain of a SePS instance, the Key is a unique identifier of the Lifecycle to which a transaction belongs. By default a TransactionKeyType complex element is populated and returned by the SePS. For targeted data retrieval functions such as CheckIntegrity, RetrieveResults, and selected RetrieveSummary scenarios, provision of the `Locator`, `Key`, and `Sequence` elements is required.

```
<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 SePS instance identifier. This commences with the ISO 3166-1 two-character country code (`CountryCode`) of the service provider followed by the `Version` number of the SePS Interface Specification (e.g. 1.15). These can be followed by `ServiceProvider` and/or

`Environment`. The first of these identifies the particular service provider and is required if the service provider is other than the postal service to which has been allocated UPU 2-character S31 issuer code corresponding to the `CountryCode` concerned. `Environment` is used in cases in which a single service provider supports more than one instance of the same version of the SePS in a given country (e.g. Production, Pilot, Training, etc …).

NOTE ISO 3166-1 country codes can be obtained at http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html#af.

EXAMPLE Royal Mail has been allocated S31 issuer code GB, so a SePS operated by it in the U.K. would not need to include `ServiceProvider` in `Locator`; a U.K.-based SePS operated by any other service provider would need to explicitly specify `ServiceProvider`.

`Locator` is returned by the SePS as part of the `TransactionKey` identifier. Callers do not use this element to indicate a particular SePS instance, that is accomplished through the SePS's SOAP URL identifier, and the use of `ServiceProvider` and/or `Environment` sub-elements does not obviate the need for a discrete URL for each specific SePS instance that callers might wish to access. The `Locator` nevertheless plays a role in crossborder SePS 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 in which the PostMark was issued regardless of which ServiceProvider operates the service. If the post has delegated SePS hosting and facilities management to more than one service provider, then multiple unique Locator elements would exist and would each bear the name of the `ServiceProvider` providing the service.

Key – A string element representing the unique transaction identifier. This `Key` is originally generated by the SePS and returned as part of the `TransactionKey`. This TransactionKey provides the SePS 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: 1) the caller is performing a RetrieveResults, RetrieveSummary, or CheckIntegrity operation (in which case the target Key of the transaction concerned is required), or 2) wishes to extend the transaction Lifecycle (See Lifecycle management in 5.5 for more information).

Sequence – A string element uniquely identifying a particular event within a multi-event business transaction Lifecycle (see Lifecycle management in 5.5). The first transaction in each Lifecycle has `Sequence` '1', even if it is the only transaction in the Lifecycle. For each subsequent operation in the Lifecycle, identified by use of the same combination of `Locator` and `Key`, `Sequence` is incremented by one so that consecutive transactions form a 'sequence' of operations. The `Sequence` element is always returned as part of the complex `TransactionKey` identifier. It can be set to null in the operation request, except when performing a RetrieveResult, or a CheckIntegrity.

7.17 TransactionStatus and TransactionStatusDetailType

For consistent error response handling, SePS-enabled applications are expected to check the overall operation status returned in the `TransactionStatus` element. This should contain a value of 0 for success, 1 for a warning or a specific error number related to the error encountered. More detailed error information is returned when available in the `TransactionStatusDetail` complex element.

EXAMPLE if a database error occurs while writing to the non-repudiation log, the `TransactionStatus` might contain 5000 and the `ErrorNumber in TransactionStatusDetail` could contain the ORA- specific error number, with `ErrorMessage` containing 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, it is intended that future versions of this specification will standardize handleable SePS ErrorNumber's and their associated ranges. Pending coverage of this topic in the specification, users are recommended to consult the interim draft, which is enitled SePS standardized error numbering scheme V1.0. Please consult the local postal service for details and availability.

7.18 ValidOperation

ValidOperation defines the list of operations supported by a SePS. The schema below lists all operations defined in this specification as valid. A SePS which only supported a subset of the defined operations would enumerate only the supported operations. 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"/>
        <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>
```

7.19 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>
```

7.20 Version

This mandatory string element shall be present on every SePS 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 SePS Interface Specification schema applicable to the formatting of the request. It is a string element and for example might contain: 1.14, 1.15, 2.0, etc… Posts may use this element in conjunction with the URL of the SePS itself to identify the version of the SePS Interface that this call is formatted under.

7.21 X509InfoType

This complex element contains several sub-elements reflecting information from the certificate used in the particular SePS operation performed. It contains a concatenation of signing or encryption certificate information.

It 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 SePS. 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 date 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 implementation-specific and represents the evidence provided by the SePS 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 implementor sees 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 a SePS implementation is not using an OCSP responder, then the post concerned is free to redefine its 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 information 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/octet-stream` when it is an ASN.1 binary OCSPBasicResponse as per RFC 2560.

```
<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>
```

8 Detailed specification of SePS operations

8.1 Introduction

This clause provides an alphabetically ordered specification of the basic operations that can be combined to implement a SePS which complies with this specification. Each sub-clause defines one such operation. A specific service will generally support only the subset of these operations which is required to provide the functionality which is relevant to the definition of the service concerned.

EXAMPLE Part B of this standard defines the EPCM service. This requires support for five core operations (CheckIntegrity, PostMark, RetrieveResults, Sign and Verify), with support for other operations being considered as extended service options. Further parts of the standard will, when developed, define other standardised service offerings, including a registered electronic mail service.

8.2 CheckIntegrity

8.2.1 CheckIntegrity Edit Rules Summary

The CheckIntegrity operation allows clients to pass in content from a previous Verify, PostMark or Sign operation and have that content byte by byte compared against the original version stored in the SePS's non-repudiation log under the requested TransactionKey. The SePS will thus validate whether the supplied 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. This prevents denial of receipt by the intended recipient provided that 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 the content stored in the SePS'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. Since `Data` is only supplied for PostMarks of Type 1, Type 2, or Type 3, the operation is only valid for previous Postmarks of these types. CheckIntegrity operations with this MimeType on PostMarks of Types 4 and 5 are not valid and should be rejected with error indication.

Please refer to the `MimeType` attribute description in 8.7.3 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 with 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 7.3), then the sender can enjoy Proof-of-Delivery, Proof-of-Possession, and Non-Repudiation of receipt. 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 7.7 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 SePS implementer to check each item and to provide the authenticity results by comparing each item to the content stored in the SePS nonrepudiation log. The CheckIntegrity operation does not perform a cryptographic verification but rather simply compares the `OriginalContent` passed in with that which already exists within the non-repudiation database under the requested `TransactionKey`.

The CheckIntegrity operation requires a valid transaction key (see TransactionKeyType in 7.16) plus an exact copy of the content 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 SePS 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 SePS decrypt the contents before comparing it.

8.2.2 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. This option is used to preserve confidentiality when using the CheckIntegrity function. The SePS will first decrypt the incoming content and compare this decrypted content to the original. The user should encrypt the data to be verified with the public key counterpart of the SePS's private decryption key and place the result in the `OriginalContent` field. The service will treat the content according to the `OriginalContentType` specified. 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`.

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 public encryption key used shall be the postal service public key.

StoreNonRepudiationEvidence – Set to 'True' to turn on database logging of non-repudiation evidence. On a CheckIntegrity, logging of information in the non-repudiation database might be required 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 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 in order to limit access to the intended recipients.

IssuePostMarkedReceipt – Set to `True` in order to cause the SePS 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 7.3. The returned receipt uses a SePS-generated timestamptoken (see 7.11 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 SePS will create the `PostMarkedReceipt` signature over the `OriginalContent` occurrences within the request. The `embedded` option of the `IssuePostMarkedReceipt` is not supported for CheckIntegrity.

Since a CheckIntegrity is inherently associated with the target TransactionKey against which it is operating, SePS 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 the CheckIntegrity should be assigned transaction key 1234 with sequence 2. If the target operation is already participating in a Lifecycle, then the SePS implementation shall retain the transaction key and increment the sequence number by 1 for the CheckIntegrity 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 open.

The `DataMimeType` element within the `Receipt` structure of the `PostMarkedReceipt` returned on a CheckIntegrity shall be set to `text/plain`.

8.2.3 CheckIntegrity Request Elements

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

```
    <xs:element name="CheckIntegrityRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:elementname="CheckIntegrityOptions"
type="epm:CheckIntegrityOptionsType"/>
                <xs:element name="SignatureType" type="epm:ValidSignatureType"
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 7.16.

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 SePS'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 7.9 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 7.3. 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 SePS implementation by re-deriving the hash over the request's `OriginalContent` element, comparing this rederived 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. SePS 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 8.12 StartLifeCycle.

SignatureType – Specifies the signature type to be used by the SePS 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 SePS.

ClientApplication – See ClientApplication in 7.4.

8.2.4 CheckIntegrity Response Object

A CheckIntegrityResponseType complex element is populated and returned by the SePS. 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 7.17)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the SePS. Please refer to TransactionKeyType in 7.16.

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 PKCS7 signature was created and subsequently verified. This plain text string was optionally returned on the target PKCS7 Verify call in the `SignedContent` element of `SignatureInfo`. It is the eContent of the PKCS7 signature and will be compared against that which was stored in the non-repudiation log as a result of the PKCS7 Verify operation.
	application/octet-stream	Same as above except that the PKCS7 `SignedContent` tobe compared is binary.
	application/vnd.upu-digest-value	Caller shall initialize the `OriginalContent` element with the `ContentHash` produced as a result of PKCS7 signature creation. This hash value was optionally returned on the PKCS7 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 PKCS7 Verify operation. NOTE 1 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 PKCS7 or XMLDSIG Verify call. This token would optionally be inside the `TimeStampToken` element of the `PostMarkedReceipt` originally returned on the target PKCS7 or XMLDSIG Verify operation. The original PKCS7 or XMLDSIG 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 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 PKCS7 Sign operation.
	text/xml	Caller shall initialize the `OriginalContent` element with the contents of the `<dsig:SignatureValue>` within the XMLDSIG `<dsig:Signature>` structure originally Signed, without the bounding tags. This value will be compared against the equivalent element from the target XMLDSIG Sign operation. NOTE 2 Do not base64 encode the `SignatureValue` twice when initializing `OriginalContent`.

Target Operation being checked	`OriginalContent's MimeType`	Processing Rules for `OriginalContent`
Sign	**text/plain**	Caller shall initialize the `OriginalContent` element with the `SignedContent` over which the original PKCS7 signature was created. This plain text string was passed in to be signed by the original PKCS7 Sign operation. It is the eContent of the PKCS7 signature and will be compared against that which was stored in the non-repudiation log as a result of the PKCS7 Sign operation.
	application/octet-stream	Same as above except that the PKCS7 `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 PKCS7 signature creation. This hash value was optionally returned on the PKCS7 Sign call in the `ContentHash` element of `SignatureInfo.` This hash value will be compared against that which was stored in the nonrepudiation log as a result of the PKCS7 Sign operation. NOTE 3 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 PKCS7 or XMLDSIG Sign call. This token would optionally be inside the `TimeStampToken` element of the `PostMarkedReceipt` originally returned on the target PKCS7 or XMLDSIG Sign operation. The original PKCS7 or XMLDSIG 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 PKCS7 Sign operation.
	text/xml	Caller shall initialize the `OriginalContent` element with the contents of the `<dsig:SignatureValue>` within the XMLDSIG `<dsig:Signature>` structure originally Signed, without the bounding tags. This value will be compared against the equivalent element from the target XMLDSIG Sign operation. NOTE 4 Do not base64 encode the `SignatureValue` twice when initializing `OriginalContent.`

Target Operation being checked	OriginalContent's MimeType	Processing Rules for 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.` The original target PostMark may be either a PKCS7 or an XMLDSIG based PostMark.
	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.` The original target PostMark may be either a PKCS7 or an XMLDSIG based PostMark.
	application/vnd.upu-digest-value	Not a valid MimeType for a CheckIntegrity against a previous PostMark. Use the `application/ timestamptoken MimeType` below for token-related comparisons.
	application/timestamp-token	Caller shall initialize the `OriginalContent` element with the detached binary RFC 3161 `TimeStampToken` optionally returned on the PostMark call. This token would be optionally inside the `TimeStampToken` element of the `PostMarkedReceipt` originally returned on the target PostMark operation. The original target PostMark may be either a PKCS7 or an XMLDSIG based PostMark.
	application/pkcs7-signature	Not a valid MimeType for a CheckIntegrity against a previous PostMark. Use the `application/ timestamptoken` for signature comparisons.
	text/xml	Caller shall initialize the `OriginalContent` element with the contents of the `dsig:SignatureValue(s)` within the `dsig:Signature` structure of the original XMLDSIG PostMark, without the bounding tags. This value will be compared against the equivalent element from the target XMLDSIG PostMark operation. NOTE 5 Do not base64 encode the `SignatureValue` twice when initializing `OriginalContent.`

8.3 Decrypt

8.3.1 Decrypt Edit Rules Summary

The Decrypt operation will decrypt any incoming `EnvelopedData` object which is encrypted using a symmetric key which is in turn encrypted using the public key counterpart of the SePS's private decryption key. Decrypt first decrypts the enveloped data's symmetric key and then uses this to decrypt the original message content, returning this in its original content format. It requires a valid `EnvelopedData` structure as an input parameter.

This can be either a PKCS7 EnvelopedData ASN.1 binary object or an XML Encryption compliant XML `EnvelopedData` node. Decrypt is most often used in Delegated Confidentiality scenarios as described in 6.4.2 Delegated Confidentiality Service. Please also refer to 7.7 EncryptResponse Option which covers the use of this option and provides examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

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

8.3.2 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 5.5 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction Lifecycle. (See Lifecycle management in 5.5 for more information.)

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. This option instructs the SePS to encrypt the `Data` element before returning it using the public key present in the incoming `RequesterSignature`. Please also refer to 7.7 EncryptResponse Option whichcovers 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 7.14 SignatureInfoType for details.

ReturnX509Info – a Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to 7.21 X509InfoType for details.

8.3.3 Decrypt Request Elements

Set the following parameters or elements of the DecryptRequestType complex element before invoking the SePS 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 7.16.

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 7.3.

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 SePS.

ClientApplication – See ClientApplication in 7.4

ContentIdentifier – See ContentIdentifier in 7.5

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 SePS will determine the envelope format from the `MimeType` attribute. The SePS will look for a value of `text/xml`, in which case the SePS will assume that the caller is passing in an XMLENC-formatted envelope, otherwise the SePS will assume the caller is passing in an ASN.1 binary CMS/PKCS7 EnvelopedData structure.

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.

8.3.4 Decrypt Response Object

A DecryptResponseType complex element is populated and returned by the SePS. 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 7.17.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the SePS. Please refer to TransactionKeyType in 7.16.

Data – A binary element containing the decrypted content. Its `MimeType` attribute will be set to `application/octet-stream` if the incoming structure was CMS/PKCS7 or it will be set to `text/xml` if the incoming structure was an XMLENC envelope. If the `EncryptResponse` option was set true, this element will contain an `EnvelopedData` object encrypted for the caller in the appropriate format.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType. Please refer to 7.14 SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to 7.21 X509InfoType for details.

8.4 Encrypt

8.4.1 Encrypt Edit Rules Summary

The Encrypt operation encrypts and returns an enveloped data object of the desired format. It will encrypt any incoming data with a certificate specified by the client, and return the encrypted envelope. The data are encrypted using a randomly chosen symmetric key (e.g. 3DES) which is subsequently encrypted (RSA key transport) using the recipient’s public key, then formatted and returned as either a PKCS7 enveloped data object or an XML Encryption compliant XML `EncryptedData` node.

The Encrypt operation provides the SePS’s native confidentiality support. The SePS 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 with whom they are dealing, including their partners or customers in an eBusiness scenario.

A special case optional usage of the SePS confidentiality capability, that works in conjunction with a local encrypt operation issued by an individual client customer using the SePS’s public encryption key, is termed “Delegated Confidentiality” within the SePS context. Use of this frees individuals from having to manage the public keys of intended recipients. A subscriber wishing to encrypt content for confidentiality reasons simply encrypts that content locally with a public key provided to them by the postal service. 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. Such local encryption using the post-specific public key can be used with any SePS 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.

The recipient does not have the private key required to decrypt the received envelope and needs to ask the post's SePS to Decrypt it for them. It would of course not make sense for the SePS to simply Decrypt the content and pass it back to the caller in the clear. Subject to the `EncryptResponse` option being turned on in the recipient's Decrypt (or Verify) request, the SePS therefore encrypts the response using the recipient's public key. To provide the SePS with the required key, the user (recipient in this case) 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 occur 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 SePS. Thusage second case, referred to as "Sign for Pickup", 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 7.7 EncryptResponse Option for a detailed description of the use of this option with examples of usage in the case of Verify, Decrypt, and RetrieveResults operations.

8.4.2 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 5.5 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction Lifecycle. (See Lifecycle management in 5.5 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 7.21 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 SePS to return a receipt attesting to the validity and non-repudiability of the operation. The digest or hash of the `Data` element passed in on the request is used to create the 3rd Reference in the PostMarkedReceipt signature (XMLDSIG) or the TimeStampToken (PKCS7). Please refer to 7.11 entitled PostMarkedReceipt for details.

If a `PostMarkedReceipt` is issued for this operation, the `DataMimeType` element within the `Receipt` structure of the `PostMarkedReceipt` will simply echo the `MimeType` passed in on the `Data` element.

DecryptIncomingEnvelope – Set to 'True' to instruct the SePS 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 SePS's private decryption key. This option is used to ensure confidential delivery of the transaction content to the SePS prior to being actually encrypted for the final intended recipient. See also `CertificateID` below. Once the `Data` has been received by the SePS and decrypted it can then be encrypted for its final intended recipient(s). When used in this fashion, the SePS is essentially performing a "Re-Encrypt".

If a `PostMarkedReceipt` is issued for this operation (see above), the `PostMarkedReceipt's` hash calculation will be over the `Data` element **after** decryption.

ReturnSignatureInfo – Set to 'True' to return a detailed breakdown of the signature object, including the original content. Please refer to 7.14 SignatureInfoType for details.

ReturnX509Info – a Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to 7.21 X509InfoType for details.

8.4.3 Encrypt Request Elements

Set the following parameters or elements of the EncryptRequestType complex element before invoking the SePS 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="epm:ValidSignatureType"/>
                <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 7.16.

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 7.3.

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 SePS.

ClientApplication – See ClientApplication 7.4

Data – The `Data` element is of Type `QualifiedDataType` and hence is always base64 encoded for transport. The `MimeType` attribute is used to tell the SePS that after decoding, this `Data` element contains a specific `MimeType` and instructs the SePS 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. 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 SePS 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 – SePS 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-xmlenccore-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.

8.4.4 Encrypt Response Object

A EncryptResponseType complex element is populated and returned by the SePS. 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 7.17)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the SePS. Please refer to TransactionKeyType in 7.16.

PostMarkedReceipt – An XML structure optionally containing the TSA-generated timestamptoken. This structure acts as the receipt. This receipt is attestation that the PostMark has been created. Please refer to 7.11 entitled PostMarkedReceipt for details.

SignatureData –This element will contain the generated PKCS7 or XMLDSIG based `EnvelopedData` structure. Its `MimeType` will be set to `application/octet-stream` for PKCS7 and `text/xml` for XMLDSIG.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType. Please refer to 7.14 SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to 7.21 X509InfoType for details.

8.5 Locate

8.5.1 Locate Edit Rules Summary

The Locate operation will retrieve an encryption certificate based on the `CertificateSearchType` and `CertificateID` specified in the supplied `LocateRequestType` complex element. This 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.

8.5.2 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 "LocateOptionsType">
        <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 5.5 for more information.)

ExtendLifecycle – Set to ‘True’ to extend the transaction Lifecycle. (See Lifecycle management in 5.5 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 7.21 X509InfoType for details.

ReturnX509Info – Set to ‘True’ to return a detailed breakdown of the certificate used to perform the operation. Please refer to 7.21 X509InfoType for details.

8.5.3 Locate Request Elements

Set the following parameters or elements of the LocateRequestType complex element before invoking the SePS 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 7.16.

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 7.3.

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 SePS.

ClientApplication – See ClientApplication in 7.4

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.

8.5.4 Locate Response Object

A LocateResponseType complex element is populated and returned by the SePS. 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 7.17)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the SePS. Please refer to TransactionKeyType in 7.16.

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 7.21 X509InfoType for details.

8.6 LogEvent

8.6.1 LogEvent Edit Rules Summary

The LogEvent operation allows customers to log and ***system*** timestamp any content they believe is of significance to the business workflow or life cycle. It is also useful when the SePS is participating with other system components which are supporting other events within the Lifecycle being logged. Examples include:

— a Single-Sign-On service, delegated under the Signature Policy to be responsible and accountable for client authentication prior to SePS consumption, which works in association with the SePS itself;

— non-repudiation of transport which is provided by system component outside the SePS. The transport service would send a LogEvent attesting to successful delivery of a document. Working in conjunction with this transport service the SePS would be able to provide the complete non-repudiation story from origin, though submission and delivery and concluding with receipt;

— support for extremely large payloads which would be inefficient to cryptographically process. In this scenario the client can hash to content and sign the hash. The signature Verify event can be sent up to the SePS as the first event in a Lifecycle, with the second event being a LogEvent covering the content itself.

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 SePS a Verify and Postmark request to close the loop in a binding way.

8.6.2 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 5.5 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction Lifecycle. (See Lifecycle management in 5.5 for more information.)

DecryptIncomingEnvelope – Set to 'True' to instruct the SePS 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 SePS's private decryption key. This option is used to ensure confidential delivery of the transaction content to the SePS. Please also refer to 7.7 EncryptResponse Option for more details.

NOTE The StoreNonRepudiationEvidence options flag is implicitly set for LogEvent operations and has therefore been intentionally omitted.

8.6.3 LogEvent Request Elements

Set the following parameters or elements of the LogEventRequestType complex element before invoking the SePS 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 7.16.

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 7.3.

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 SePS.

ClientApplication – See ClientApplication in 7.4.

ContentIdentifier – See ContentIdentifier in 7.5.

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

ContentMetadata – See ContentMetadata

8.6.4 LogEvent Response Object

A LogEventResponseType complex element is populated and returned by the SePS. 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 eleme

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the SePS. Please refer to TransactionKeyType in 7.16.

8.7 PostMark

8.7.1 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 five content types described below and specified in the `MimeType` attribute of the `Data` input request element. As part of the PostMark operation, the SePS will timestamp the content through its TSA or equivalent component and return, in the `PostMarkedReceipt` sub-element of the response, an RFC 3161-compliant timestamptoken in the `TimeStampToken` element, a `Receipt` structure, and a `ReceiptSignature` structure. See 7.11 PostMarkedReceipt for details.

8.7.2 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 5.5 for more information.)

ExtendLifecycle – Set to ‘True’ to extend the transaction Lifecycle. (See Lifecycle management in 5.5 for more information.)

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

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. When ‘True’, it instructs the SePS to encrypt the `SignatureData` element, using the public key present in the incoming `RequesterSignature,` before returning it. See 7.7 EncryptResponse Option for a description of 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.

8.7.3 Postmark Request Elements

Set the following parameters or elements in the PostMarkRequestType complex element before invoking the SePS as appropriate. The PostMarkImage element allows callers to request that an image be returned. This is consistent with the Verify operation.

```
    <xs:element name="PostMarkRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="PostMarkOptions" type="epm:PostMarkOptionsType"/>
                <xs:element name="SignatureType" type="epm:ValidSignatureType"/>
                <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="PostMarkImage" type="epm:PostMarkImageType"
nillable="true" minOccurs="0"/>
                <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` (see 8.7.2).

SignatureType – Specifies the signature type to be used by the SePS 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 a Lifecycle is being extended. Please refer to TransactionKeyType in 7.16.

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 7.3).

OrganizationID – A string element which identifies the organization requesting the service. This value shall match against a list of valid Organization Identifiers that are registered with the SePS.

ClientApplication – See ClientApplication in 7.4

ContentIdentifier – See ContentIdentifier in 7.5

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 to be PostMarked. For both SignatureType values of PKCS7 and XMDLSIG, this data will be hashed (after decoding), for the caller's convenience prior to internally calling the SePS's timestamping component. For `SignatureType` PKCS7 a timestamptoken, will be returned in the `TimeStampToken` element of the `PostMarkedReceipt`. When the `SignatureType` is XMLDSIG the `PostMarkedReceipt` signature will be over the hash of the content. Please refer to Example 2 – Standalone <PostMarkedReceipt> over Data when using PostMark operation.

— ***Type 2*** "application/octet-stream" should be used when passing in binary data to be PostMarked. This data will also be hashed (after decoding), for the caller's convenience prior to internally calling the SePS's timestamping component. For `SignatureType` PKCS7 a timestamptoken, will be returned in the `TimeStampToken` element of the `PostMarkedReceipt`. When the `SignatureType` is XMLDSIG the `PostMarkedReceipt` signature will be over the hash of the content. Please refer to Example 2 – Standalone <PostMarkedReceipt> over Data when using PostMark operation.

— ***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 caller will be used to construct the call to the SePS's 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 (e.g. sha256, sha512). For `SignatureType` PKCS7, a `TimeStampToken`will be returned in the `TimeStampToken` element of the `PostMarkedReceipt`. When the `SignatureType` is XMLDSIG the `PostMarkedReceipt` signature will be over the already hashed content which is the digest value passed in on the request.

— ***Type 4*** "application/pkcs7-signature" which is PKCS7 SignatureType-specific, indicates that the required timestamp is to be added to the incoming PKCS7 signature structure. The timestamptoken's message imprint 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 standalone `PostMarkedReceipt` structure will also be returned. This is consistent with Verify `IssuePostMarkedReceipt` processing.

— ***Type 5*** "text/xml", which is XMLDSIG SignatureType-specific, indicates that the required `PostMarkedReceipt` is to be calculated over the incoming XMLDSIG-based signature structure. The `PostMarkedReceipt` signature's 3rd Reference will reference a copy of the `SignatureValue` element(s) extracted from the incoming signature(s), and will be returned in the `PostMarkedReceipt` response element.

PostMarkImage – Please refer to the identically named section in 7.11

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.

8.7.4 PostMark Response Object

A PostMarkResponseType complex element containing the following elements is populated and returned by the SePS:

```
    <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' that a warning was generated. Transaction failure is denoted by a numeric value, other than 0 or 1, which specifies the type of error.

TransactionStatusDetail – an element whose type is a complex element defined by TransactionStatusDetailType (see TransactionStatus and TransactionStatusDetailType in 7.17).

TransactionKey – an element whose type is a complex element defined by TransactionKeyType (see TransactionKeyType in 7.16). The Locator, Key, and Sequence elements are populated and returned by the SePS.

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. See PostMarkedReceipt for details.

SignatureData – If the incoming `Data` element's `MimeType` is Type 4 which necessitates a PKCS7 PostMark, then the timestamp will be part of the updated and returned signature structure and will be returned in this element. The embedded 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 the signature at that moment in time and does not attest to the validity of anything else.

If the incoming `Data` element's `MimeType` is Type 5 which necessitates an XMLDSIG PostMark, then the`TstInfo` Reference will be part of the updated and returned XMLDSIG signature structure and will be returned in this element. The optional embedded timestamp token will be a conventional RFC 3161 binary timestamp token and will be included in the `Receipt` structure. The `PostMarkedReceipt` when using Type 5, is simply an attestation of the existence of the signature at that moment in time and does not attest to the validity of anything else.

8.8 RetrievePostalAttributes — 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 SePS 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 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 SePS-compliant desktop application. They 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 SePS's desktop appearance.

```
    <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 SePS 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="ExpiresOnDateTime" 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>
```

8.9 RetrieveResults

8.9.1 RetrieveResults Edit Rules Summary

The RetrieveResults operation is used both for accessing evidence required in the case of a challenge as to the authenticity of one or more events in a Lifecycle and to provide a document pickup service. In the first case, it is used by the post in support of 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 in respect of most operations/verbs; exceptions are RetrievePostalAttributes and RetrieveSummary, both of which are already fundamentally retrieval operations and not the subject of disputes.

In case of usage as a document pickup facility, a recipient can pickup a document originally placed with the SePS as a result of a sender locally signing a document and subsequently requesting that the SePS Verify that signed document. The act of verifying the document's signature via an origin Verify operation issued by the sender can place both the signature and the signed document (whether embedded or detached) in the SePS's database under a specific TransactionKey, which is returned to the sender. The sender can then pass the TransactionKey “out of band” to the recipient, instructing them to “pickup” the document. To do this, the recipient issues a RetrieveResults operation with the specified TransactionKey, resulting in pickup of the document. Caution should be taken as to whom is allowed to pickup a document in this fashion. If security is a concern, implementers should use a StartLifecycle specifying the `ParticipatingParties` allowed to perform actions against the content. See also `AccessScope` and `AccessLevel` in 6.12 StartLifecycle for further access privilege details.

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`. When issuing the StartLifecycle request, John could specify CN=Bill Graham, O=Acme Corporation, C=CA as a valid `ParticipatingParty` and `AccessLevel` for Bill set to `Signed`. John could also set `NotifyEvents` for Bill to `Verify` so Bill would be eMailed when John posts the document to the SePS via his Verify operation. John then signs the document and subsequently verifies that document/signature using a Verify request to the SePS. This places the verified document and signature along with all results in the SePS 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 SePS-enabled desktop applications or can be directly implemented within subscriber SePS-enabled applications. Consult the local postal service 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 the signature verification operation for which the caller wants the results. The `RequesterSignature` element of `ClaimedIdentity` shall be initialized.

To prevent unauthorized access to an operation's potentially sensitive contents, usage of RetrieveResults for an operation is, by default, restricted to the individual that initiated that operation concerned. To allow wider access to the response content of an operation, the user 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 SePS is witnessing the origin and destination events. That is, the Verify of the sender's signature over the document, and the recipient’s signed CheckIntegrity request upon receipt. The recipient is asked to sign the `OriginalContent` of the CheckIntegrity request thus ensuring Proof-of-Delivery and Proof-of-Possession.

8.9.2 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’.

```
    <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 SePS to return a receipt attesting to the validity and non-repudiability of the operation. Please refer to PostMarkedReceipt for details. 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.

The `DataMimeType` element within the `Receipt` structure of the `PostMarkedReceipt` returned on a RetrieveResults shall be set to `text/plain`.

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. When ‘True’, it instructs the SePS to encrypt the `Data` element of the `Results` structure, using the public key present in the incoming `RequesterSignature`, before returning it. If the caller has also requested that the `SignatureInfo` be returned, then the `SignedContent` element should also be encrypted.

See 7.7 EncryptResponse Option for a description of 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.

NOTE 1 The only non-repudiation evidence of significance is the `PostMarkedReceipt`. As such, the `IssuePostMarkedReceipt` option should also be set to true in order to gain any value from this option.

ReturnTimeStampAudit – Set to 'True' to request population of the TimeStampAudit element of the `RetrieveResultsResponseType` (see 8.9.4).

ReturnSignatureInfo – Set to 'True' to request return of a detailed breakdown of the signature object, including the original content. See 7.14 SignatureInfoType for details.

ReturnX509Info – Set to 'True' to request return of a detailed breakdown of the certificate used to perform the operation. See 7.21 X509InfoType for details.

NOTE 2 Since a RetrieveResults is inherently associated with the target `TransactionKey` against which it is operating, SePS 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 SePS 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 open.

8.9.3 RetrieveResults Request Elements

Set the following parameters or elements of the RetrieveResultsRequestType object type appropriately before invoking the SePS:

```
    <xs:element name="RetrieveResultsRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="RetrieveResultsOptions"
type="epm:RetrieveResultsOptionsType"/>
                <xs:element name="SignatureType" type="epm:ValidSignatureType"
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:sequence>
        </xs:complexType>
    </xs:element>
```

RetrieveResultsOptions – An element whose type is a complex element defined by `RetrieveResultsOptionsType` from 8.9.2.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. `Locator` should contain the corresponding SePS application instance reference. `Key` should contain the transaction identifier (key) from the previous target operation for which results and content are to be retrieved. `Sequence` should contain the respective sequence number of the target transaction. See TransactionKeyType in 7.16.

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 7.3)

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

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

ClientApplication – See ClientApplication in 7.4

ContentIdentifier – See ContentIdentifier in 7.5

8.9.4 RetrieveResults Response Object

A RetrieveResultsResponse complex element is populated and returned by the SePS. 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 SePS's log are populated and returned in the `ResultsType` structure. A description of all response elements follows:

```
    <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 7.17)

TransactionKey – This is the new TransactionKey generated by the SePS, 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 SePS Please refer to TransactionKeyType in 7.16.

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 1 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 ref="epm:PostMarkedReceipt" minOccurs="0"/>
            <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:element name="SignatureType" type="epm:ValidSignatureType"
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 7.17

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 8.10 for details)

ClientApplication – See ClientApplication in 7.4

ContentIdentifier – See ContentIdentifier in 7.5

PostMarkedReceipt – This is the `PostMarkedReceipt`, if any, optionally returned for the original target operation being retrieved. For example a `PostMarkedReceipt` may have been returned on a Verify operation if the `IssuePostMarkedReceipt` option was selected. This applies to standalone `PostMarkedReceipts` only. Please refer to 7.11 for further details.

Data – This is an unbounded element whose first occurrence is always the serialized SOAP request structure of the target operation being retrieved. Its `MimeType` will be set to `application/soap+xml`. The contents of the second occurrence of the `Data` element is the primary response 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 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 second occurrence of 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 target operation.

Original Operation	Contents of 2nd `Data element occurence`	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	**Nill**	
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 2 The `SignedContent` element from the original Sign request is also available in the `SignatureInfo` element of the `ResultsType`.*
StartLifeCycle	**Nill**	
PostMark	**Nill**	
Verify	**Nill or SignatureData**	Will be Nill unless the original Verify operation requested a `PostMarkedReceipt` whose `Location` was `embedded.` In this case the 2nd `Data` element will contain the updated and returned `SignatureData` element. *NOTE 3 The `SignedContent` element from the original Verify request is also available in the `SignatureInfo` element.*

NOTE 2 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 SePS implementation.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType. Please refer to 7.14 entitled SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to 7.21 entitled X509InfoType for details.

8.10 RetrieveSummary

8.10.1 RetrieveSummary Edit Rules Summary

The RetrieveSummary operation can be used for two main purposes:

— to access a summary list of all the events that have taken place in the Lifecycle designated by a given TransactionKey;

— to access all the Transaction Keys, for a given OrganizationID, which satisfy supplied selection criteria. Once the desired event(s) have been located from the returned list of `TransactionKey`'s that meet the criteria, the client can access their specific details using the RetrieveResults operation outlined in 8.8.

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. Supply of the `RequesterSignature` element of `ClaimedIdentity` is required.

8.10.2 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 5.5 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction Lifecycle. (See Lifecycle management in 5.5 for more information.)

8.10.3 RetrieveSummary Request Elements

Set the following parameters or elements of the RetrieveSummaryRequestType complex element before invoking the SePS 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 7.16.

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 7.3.

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 SePS.

ClientApplication – See ClientApplication in 7.4

ContentIdentifier – See ContentIdentifier in 7.5

LastUniqueSequenceId – This element allows an application to retrieve transactions which satisfy the selection criteria after a particular sequence number within the SePS'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.

8.10.4 RetrieveSummary Response Object

A RetrieveSummaryResponseType complex element is populated and returned by the SePS. 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>
```

8.11 Sign

8.11.1 Sign Edit Rules Summary

The Sign operation is a server-side Sign and not the Sign performed by the client endpoint (normally a customer or ISV application). Sign is useful in two main use-cases. The first such use-case applies when subscribing organizations, wishing to ensure their partners of the authenticity of the content they receive, 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 SePS or is delegated to the enterprise or an Identity Service Provider working in conjunction with the customer and the postal service. This alleviates the postal service 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 SePS for verification and PostMarking using the SePS’s Verify operation.

The Sign operation requires provision of the content to be signed and returns its result 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.

SePS 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 SePS 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, SePS implementations should honor or reject requests based on their capability.

8.11.2 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 5.5 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction Lifecycle. (See Lifecycle management in 5.5 for more information.)

VerifyCertificate – Set to 'True' to enable certificate status checking. 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 request OCSP or CRL checking of the revocation status of the certificate to be used for the signing operation.

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 SePS to return a receipt attesting to the validity and non-repudiability of the operation. This returned receipt is PostMarked using a SePS-generated timestamptoken. Please refer to 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. The `DataMimeType` element within the `Receipt` structure of the `PostMarkedReceipt` returned on a Sign shall be set to `application/pkcs7-signature` (PKCS7) or `text/xml` (XMLDSIG) since the `PostMarkedReceipt` is over the `SignatureValue` field/element.

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

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. When 'True', it i instructs the SePS to encrypt the `SignatureData` element, using the public key present in the incoming `RequesterSignature`, before returning it. If the caller has also requested that the `SignatureInfo` be returned, then the `SignedContent` sub-element should also be encrypted. Refer to 7.7 EncryptResponse Option for a description of the use of this option with examples of usage in 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 7.14 SignatureInfoType for details.

ReturnX509Info – Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to 7.21 X509InfoType for details.

8.11.3 Sign Request Elements

Set the following parameters or elements of the SignRequestType complex element before invoking the SePS 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="epm:ValidSignatureType"/>
                <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 7.16.

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 7.3.

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 SePS.

ClientApplication – See ClientApplication in 7.4

ContentIdentifier – See ContentIdentifier in 7.5

Data – This element contains the content to be signed and is by definition base64-encoded for transport as dictated bt its `QualifiedDataType`. The MimeType attribute is used to tell the SePS that after base64 decoding, this `Data` element contains a specific `MimeType` and instructs the SePS how to format the input `Data` in the resulting signature. Valid values for `MimeType` are `text/xml`, `text/plain`, or `application/octetstream`.

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. This is especially true for XMLDSIG-based signatures. `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, XMLDSIG-detached-external,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 same-document detached signature should specify XMLDSIG-detached. XMLDSIG-detached-external is specified when the signature references signed content which is external to or outside of the signature altogether. Specifying a value of XMLDSIG-template instructs the SePS 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 C for specific examples of signing templates.

KeyName – This optional element is used in server-side signing scenarios where a SePS 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 SePS implementation to access the private key for this sign operation. SePS implementations are free to decide how authentication and the password is utilized in serverside 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 SePS 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 7.15 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.

8.11.4 Sign Response Object

A SignResponseType complex element is populated and returned by the SePS. 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 7.17) TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the SePS. Please refer to TransactionKeyType in 7.16.

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 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 PostMarkedReceipt for more details.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType Please refer to 7.14 SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to 7.21 X509InfoType for details.

8.12 StartLifeCycle

8.12.1 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 5.5 for more information). The operation requires either a valid participating party group (refer to ParticipatingPartyType in 7.10 for details) unless the `AccessScope` element is set to `Global` (see 7.2 AccessScope and Scopes). This will restrict who can access, participate in, and contribute to the Lifecycle.

The StartLifecycle operation is an optional operation. It needs to be explicit only 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: 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.

8.12.2 StartLifecycleOptions Request Flags

None – A StartLifeCycleOptionsType element does not exist.

8.12.3 StartLifecycle Request Elements

Set the following parameters or elements of the StartLifeCycleRequestType complex element before invoking the SePS 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 7.3.

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 SePS.

ClientApplication – See ClientApplication in 7.4

ParticipatingParty – An element whose type is a complex element defined by ParticipatingPartyType. (refer to ParticipatingPartyType in 7.10 for details)

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

8.12.4 StartLifecycle Response Object

A StartLifeCycleResponseType complex element is populated and returned by the SePS. 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 7.17)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the SePS. Please refer to TransactionKeyType in 7.16.

8.13 Verify

8.13.1 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 this is explicitly 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 SePS 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.

8.13.2 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 5.5 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction Lifecycle. (See Lifecycle management in 5.5 for more information.)

VerifyCertificate – Set to 'True' to enable certificate status checking. If set to 'False', 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. Refer to 7.21 X509InfoType for details.

DecryptIncomingEnvelope – Set to 'True' to instruct the SePS to decrypt `SignedContent` and `SignatureData` before performing verification. `SignedContent` and `SignatureData`, included as EnvelopedData objects, shall be encrypted with the public key counterpart of the SePS's private decryption key. This option is used to ensure confidential delivery of the transaction content to the SePS. See also "EncryptResponse" below.

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. When 'True', it instructs the SePS to encrypt the `SignatureData` element, using the public key present in the incoming `RequesterSignature`, before returning it. If the caller has also requested that the `SignatureInfo` be returned, then the `SignedContent` sub-element should also be encrypted. Refer to 7.7 EncryptResponse Option for a description of the use of this option with examples of usage in 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 SePS to return a receipt attesting to the validity and non-repudiability of the Verify operation. The returned receipt is PostMarked using a SePS-generated timestamp. See PostMarkedReceipt for details. The timestamp and other receipt information is returned in a `PostMarkedReceipt` element which is itself bound with a signature of authenticity. The `Location` sub-element,which has valid values `standalone` and `embedded`, instructs the SePS where to place the resulting `PostMarkedReceipt` element. For PKCS7 and XMLDSIG based signatures, a value of `standalone` instructs the SePS to return a separate standalone `PostMarkedReceipt` as shown in the example in Annex C.2 Standalone PostMarkedReceipt over a verified signature. For XMLDSIG signatures, a value of `embedded` instructs the SePS to embed a `PostMarkedReceipt` structure into the incoming signature upon successful verification as shown in Annex C.4 Embedded <PostMarkedReceipt> over a verified signature. The updated signed document, which is now also PostMarked, is returned in the `SignatureData` element of the verify response.

If the incoming signed document to be verified already contains an embedded PostMarkedReceipt, and the Verify request is asking for another PostMarkedReceipt by means of the IssuePostMarkedReceipt request option, SePS implementations should not replace the current embedded PostMarkedReceipt. Instead they shall leave the existing PostMarkedReceipt intact and return a standalone PostMarkedReceipt. This will allow multiple PostMarkedReceipts to be assigned to a document over time and across changes.

If the incoming signed document to be verified already contains a PostMarkedReceipt, and the Verify request is asking for another PostMarkedReceipt by means of the IssuePostMarkedReceipt request option, SePS implementations shall not replace the current embedded PostMarkedReceipt. Instead they shall leave the existing PostMarkedReceipt intact and return a standalone PostMarkedReceipt. This will allow multiple PostMarkedReceipts to be assigned to a document over time and across changes.

The `DataMimeType` element within the `Receipt` structure of the `PostMarkedReceipt` returned on a Verify shall be set to `application/pkcs7-signature` (PKCS7) or `text/xml` (XMLDSIG).

For PKCS7 signatures, a value of `embedded` instructs the SePS 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. When applied to the verification of PKCS7 signatures `embedded` thus really means *"Create and return a PostMarkedReceipt structure, and also embed an RFC 3161 timestamp token into my signature".* This extended meaning is not 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="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>
```

If an embedded timestamptoken already exists within an incoming PKCS7 signature, the SePS implementation shall 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 request return of a detailed breakdown of the signature object, including the original content. Please refer to 7.14 SignatureInfoType for details.

ReturnX509Info – Set to 'True' to request return of a detailed breakdown of the certificate used to perform the operation. Please refer to 7.21 X509InfoType for details.

8.13.3 Verify Request Elements

Set the following parameters or elements of the VerifyRequestType element before invoking the SePS 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 a Lifecycle is being extended. Please refer to TransactionKeyType in 7.16.

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 7.3.

OrganizationID – A string element which identifies the organization requesting the service. This value shall match against a list of valid Organization Identifiers that are registered with the SePS.

ClientApplication – See ClientApplication in 7.4

ContentIdentifier – See ContentIdentifier in 7.5

SignatureData – This element is 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 SePS will determine the signature format from the `MimeType` attribute (see below).

MimeType – This is an attribute of the `SignatureData` element above and shall specify `text/xml` if `SignatureData` contains an XMLDSIG-formatted signature; `application/pkcs7-signature` if it contains an ASN.1 binary PKCS7 SignedData object or `application/pdf` if a signed PDF is passed in.

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, a SePS 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 SePS.

There are 2 scenarios to be addressed by the SePS implementation when Verifying a `PostMarkedReceipt, they correspond to signature PostMarks versus Data PostMarks.` For each scenario the PKCS7 and XMLDSIG steps are described separately:

a) When Verifying a PKCS7-based `PostMarkedReceipt` which was originally created over a PKCS7 signature, the following verification steps shall be performed:

   1) Verify the PKCS7 signature contained in the `SignatureData` request element

   2) 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.

   3) 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.

b) When Verifying an XMLDSIG-based `PostMarkedReceipt` which was originally created over an XMLDSIG signature, the following verification steps shall be performed:

   1) Verify the XMLDSIG signature contained in the `SignatureData` request element

   2) Verify the XMLDSIG `PostMarkedReceipt` signature contained in the `PostMarkedReceipt` request element

   3) 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 performing an equality check comparing the `dsig:SignatureValue` of the signature in the `SignatureData` element with the `PostMarkedSignatureValue` element from the PostMarkedReceipt.

c) When Verifying a PKCS7-based `PostMarkedReceipt` which was originally created over data, as would be the case when a PostMark operation was used, the following verification steps shall be performed:

   1) Verify the `ReceiptSignature` element within the incoming `PostMarkedReceipt.` For PKCS7-based receipts, the `ReceiptSignature` itself is a detached signature over the serialized contents of the `Receipt` element.

   2) 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.

d) When Verifying an XMLDSIG-based `PostMarkedReceipt` which was originally created over data, as would be the case when a PostMark operation was used, the following verification steps shall be performed:

1) Verify the XMLDSIG `PostMarkedReceipt` signature contained in the `PostMarkedReceipt` request element

2) Perform an equality check to confirm that this `PostMarkedReceipt` is bound to the data in the incoming `SignedContent.` This is accomplished by comparing the digest of the data in the `SignedContent` element against the base64-decoded content of the `PostMarkedContent` element of the `PostMarkedReceipt`. SePS implementations shall use the algorithm specified in the `DigestAlgo` element of the `Receipt`.

NOTE In both verification scenarios above, the `MessageImprint` element of the `Receipt` can be used to alleviate ASN1 parsing of the `TimeStampToken`. Please also refer to PostMarkedReceipt for a discussion of the `MessageImprint` element.

SignedContent –This optional element is required for the verification of detached PKCS7 signatures and when Verify'ing `PostMarkedReceipt` signatures. This element should be initialized when detached PKCS7 signatures are being verifiedor when a `PostMarkedReceipt` is being verified. When Verify'ing `PostMarkedReceipts of Types 1, 2, and 3`, the `SignedContent` element is required and must be initialized by the caller.

When verifying a `PostMarkedReceipt`, of Types 1, 2, and 3 this element shall contain the data that was originally PostMarked. It is required to confirm the binding of the `PostMarkedReceipt` signature to the data that was PostMarked. For example, when Verifying an XMLDSIG PostMarkedReceipt of Type 1 which is MimeType text/plain, the caller should pass up the original data that was PostMarked. This data will be re-hashed and compared against the `PostMarkedContent` element which is the 3rd Reference in the `PostMarkedReceipt` signature.

For XMLDSIG based signatures, which may contain a complex series of References both internal to and external to the signed document, the SePS expects the use of same-document detached signatures and expects the data to be part of the signed document. Therefore the SignedContent is element need not be initialized when Verify'ing XMLDSIG-based signatures. For signatures whose References happen to be external to the signed document, the SePS assumes that the Reference's URI attribute will resolve successfully.

SignatureSelector – This element applies to XMLDSIG-based signatures only. This optional `SignatureSelector` element qualifies the XMLDSIG signature(s) to be verified by the SePS. 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 1 The user would specify string values of `lgl:Party1` and/or `lgl:Party2` to explicitly instruct the SePS what to Verify. By default the SePS 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 2 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` is 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.

8.13.4 Verify Response Object

A VerifyResponseType complex element containing the following elements is populated and returned by the SePS:

```
    <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' that a warning was generated. Transaction failure is denoted by a numeric value, other than 0 or 1, which specifies the type of error.

TransactionStatusDetail - an element whose type is a complex element defined by TransactionStatusDetailType. See TransactionStatus and TransactionStatusDetailType in 7.17)

TransactionKey - an element whose type is a complex element defined by TransactionKeyType (see TransactionKeyType in 7.16). The Locator, Key, and Sequence elements are populated and returned by the SePS.

PostMarkedReceipt - an optional PostMarkedReceipt structure which can be returned when the IssuePostMarkedReceipt option is specified. It is always returned if the Location element specifies standalone. If the Location element specifies embedded, then this element be returned, together with an embedded RFC 3161 timestamp token, in the case of PKCS7-based signatures, but will be empty in the case of XMLDSIG-based signatures as the entire receipt will be embedded in the XML signature. See

IssuePostMarkedReceipt above for more details.

SignatureData is returned if a value of embedded is specified in the Location element of the IssuePostMarkedReceipt. When verifying PKCS7-based signatures, the returned SignatureData element will contain an updated PKCS7 signature which includes an embedded signature timestamp 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. If it is specified NodeName may affect the scope of the PostMarkedReceipt signature.

ContentMetadata – any ContentMetaData element initialized by the Verify request is returned to the caller.

SignatureInfo - an element whose type is a complex element defined by SignatureInfoType. See 7.14 SignatureInfoType for details.

X509Info - an element whose type is a complex element defined by X509InfoType. See 7.21 X509InfoType for details.

The following table summarizes the valid Verify options and the elements required in each scenario. A short description is also provided.

Valid Verify Request Combinations

SignatureData (MimeType)	PostMarkedReceipt (Type)	SignedContent	Procesing Description
text/xml			Verify incoming XMLDSIG signature using XMLDSIG crypto support.
text/xml	Type 5		Verify incoming XMLDSIG signature and PostMarkedReceipt (PMR) using XMLDSIG crypto support. Perform equaliy check of SignatureValue in SignedData's signature against PostMarkedSignatureValue element within PMR.
	Type 1 or Type 2	Req'd	Verify incoming PMR using XMLDSIG crypto support. Perform equality check of hash of SignedContent against base64-decoded value in PostMarkedContent element.
	Type 3	Req'd	Verify incoming PMR using XMLDSIG crypto support. Perform equality check of SignedContent (already a hash) against base64-decoded value in PostMarkedContent element.
application/pkcs7-signature			Verify incoming PKCS7 enveloping signature using PKCS7 crypto support.
application/pkcs7-signature		Req'd	Verify incoming PKCS7 detached signature using PKCS7 crypto support.
application/pkcs7-signature	Type 4		Verify incoming PKCS7 enveloping signature using PKCS7 crypto support. Serialize PMR's Receipt elements and perform detached PKCS7 signature verify. Perform equality check of hash of PKCS1 from SignedData's signature against messageImprint from PMR's TimeStampToken.
application/pkcs7-signature	Type 4	Req'd	Verify incoming PKCS7 detached signature using PKCS7 crypto support. Serialize PMR's Receipt elements and perform detached PKCS7 signature verify. Perform equality check of hash of PKCS1 from SignedData's signature against messageImprint from PMR's TimeStampToken.
	Type 1 or Type 2	Req'd	Serialize PMR's Receipt elements and perform detached PKCS7 signature verify. Perform equality check of hash of SignedContent against messageImprint from PMR's TimeStampToken.
	Type 3	Req'd	Serialize PMR's Receipt elements and perform detached PKCS7 signature verify. Perform equality check of SignedContent (already a hash) against messageImprint from PMR's TimeStampToken.

If the `IssuePostMarkedReceipt` option is turned on for any of the above combinations a `PostMarkedReceipt` of either Type 4 (PKCS7) or Type 5 (XMLDSIG) is returned in the Response.

If the `DecryptIncomingEnvelope` option is turned on for any of the above combinations, the SePS shall decrypt either or both the `SignatureData` and `SignedContent` elements based on the specific combination in question.

Annex A
(normative)

SePS XML Schema V1.15

This Annex contains the full expanded content of version V1.15 of the SePS Interface Schema XML .xsd file, the version applicable at the date of publication of this version of the standard.

```
<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 name="XMLSignedReceipt" type="epm:QualifiedDataType"/>
                </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="DigestAlgo" type="xs:string" nillable="true"/>
            <element name="DataMimeType" type="xs:string"/>
            <element name="PostMarkImage" nillable="true">
                <complexType>
                    <simpleContent>
                        <extension base="xs:base64Binary">
                            <attribute name="Format" type="xs:string" default="JPG"/>
                            <attribute name="Size" type="epm:ValidImageSize" default="Small"/>
                        </extension>
                    </simpleContent>
                </complexType>
            </element>
            <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"/>
                <xs:element name="PostMarkImage" type="epm:PostMarkImageType" nillable="true" minOccurs="0"/>
                <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" 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"/>
            </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="ExpiresOnDateTime" 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 B
(normative)

Web Service Description Language (WSDL) V1.15

This Annex contains the full expanded content of version V1.15 of the SePS Web Services Description Language (.wsdl) file applicable as at the date of publication of this version of the standard.

```
<?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>
```

Annex C
(informative)

Examples

C.1 General

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.

C.2 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 [SePS] 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="PostMarkedReceiptSignature" 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="#TstInfo">
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>jWkUFR6epvkrtaxTiQ33DiWy+l8=</dsig:DigestValue>
        </dsig:Reference>
        <dsig:Reference URI="#Receipt">
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>9JWKdLh/8Cs9Slu2QmZixOJl+x0=</dsig:DigestValue>
        </dsig:Reference>
        <dsig:Reference URI="#PostMarkedSignatures">
            <dsig:Transforms>
                <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
            </dsig:Transforms>
            <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="urn:oasis:names:tc:dss:1.0:core:schema" Id="TstInfo">
            <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/schemas" Id="Receipt">
            <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>
                <TSAX509SubjectName> … </TSAX509SubjectName>
                <MessageImprint> … </MessageImprint>
                <DigestAlgo>sha1</DigestAlgo>
                <DataMimeType>text/plain</DataMimeType>
                <PostMarkImage> … </PostMarkImage>
                <RevocationStatusQualifier>CRL Checked</RevocationStatusQualifier>
                <TimeStampToken MimeType="application/pkcs7-signature"></TimeStampToken>
                <ReceiptMetadata>
                    <Name> … </Name>
                    <Value>… </Value>
                </ReceiptMetadata>
            </Receipt>
        </epm:PostMarkedReceipt>
    </dsig:Object>
    <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <epm:PostMarkedContent xmlns:epm="http://www.upu.int/EPMService/schemas" Id="PostMarkedSignatures">
            <epm:PostMarkedSignatureValue>1NiHC2bBKfT ... AlfhecQo=</epm:PostMarkedSignatureValue>
        </epm:PostMarkedContent>
    </dsig:Object>
</dsig:Signature>
```

If the standalone PostMarkedReceipt covers more than one signature, the 3rd Referenced Object would look like this:

```
<dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
    <epm:PostMarkedContent xmlns:epm="http://www.upu.int/EPMService/schemas" Id="PostMarkedSignatures">
        <PostMarkedSignatureValue>1NiHC2bBKfT ... AlfcGhecQo=</PostMarkedSignatureValue>
        <PostMarkedSignatureValue>aqw95gB/Tz5 ... n0qRqMHJ5c=</PostMarkedSignatureValue> ...
        ... would include as many other PostMarkedSignatureValue elements as may be present in the PostMarked document
        ...
    </epm:PostMarkedContent>
```

```
</dsig:Object>
```

C.3 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 transform specified.

```
<?xml version="1.0" encoding="UTF-8"?>
<dsig:Signature Id="PostMarkedReceiptSignature" 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="#TstInfo">
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>jWkUFR6epvkrtaxTiQ33DiWy+l8=</dsig:DigestValue>
        </dsig:Reference>
        <dsig:Reference URI="#Receipt">
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>9JWKdLh/8Cs9Slu2QmZixOJl+x0=</dsig:DigestValue>
        </dsig:Reference>
        <dsig:Reference URI="#PostMarkedData">
            <dsig:Transforms>
                <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
            </dsig:Transforms> <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="urn:oasis:names:tc:dss:1.0:core:schema" Id="TstInfo">
            <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/schemas" Id="Receipt">
            <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>
                <TSAX509SubjectName> … </TSAX509SubjectName>
                <MessageImprint> … </MessageImprint>
                <DigestAlgo>sha1</DigestAlgo>
                <DataMimeType>text/plain</DataMimeType>
                <PostMarkImage> … </PostMarkImage>
                <RevocationStatusQualifier>Not Applicable</RevocationStatusQualifier>
                <TimeStampToken MimeType="application/pkcs7-signature"></TimeStampToken>
                <ReceiptMetadata>
                    <Name>...</Name>
                    <Value> … </Value>
                </ReceiptMetadata>
            </Receipt>
        </epm:PostMarkedReceipt>
    </dsig:Object>
    <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <epm:PostMarkedContent xmlns:epm="http://www.upu.int/EPMService/schemas" Id="PostMarkedData"
        >RGF0YSBgdG8gcQ...gVGkgMTUgMTI6MTA=</PostMarkedContent>
    </dsig:Object>
</dsig:Signature>
```

C.4 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:

1) a standard `<dss:TstInfo>` as per [DSSCore];

2) an `<epm:PostMarkedReceipt>` element from the [SePS] schema;

3) 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 Id="PostMarkedReceiptSignature" 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="#TstInfo">
                <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <dsig:DigestValue>3Lk/6TE71dqeXZFUJ9qqaPInm24=</dsig:DigestValue>
            </dsig:Reference>
            <dsig:Reference URI="#Receipt">
                <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-xpath-19991116">
                        <dsig:XPath xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                            ancestor-or-self::dsig:SignatureValue[../@Id!="PostMarkedReceiptSignature"]
                        </dsig:XPath>
                    </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# Id="TstInfo">
        <dss:TstInfo xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema">
            <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# Id="Receipt">
            <epm:PostMarkedReceipt xmlns:epm="http://www.upu.int/EPMService/schemas">
                <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>
                    <TSAX509SubjectName> … </TSAX509SubjectName>
                    <MessageImprint> … </MessageImprint>
                    <DigestAlgo>sha1</DigestAlgo>
                    <DataMimeType>text/plain</DataMimeType>
                    <PostMarkImage> … </PostMarkImage>
                    <RevocationStatusQualifier>CRL Checked</RevocationStatusQualifier>
                    <TimeStampToken MimeType="application/pkcs7-signature"></TimeStampToken>
                    <ReceiptMetadata>
                        <Name> ... </Name>
                        <Value> … </Value>
                    </ReceiptMetadata>
                </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>
```

C.5 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>
```

C.6 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 D
(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 SePS. Their influence and role in shaping the SePS 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 (ASN.1) to define the structure of the electronic signature. This structure is based on the structure defined in RFC 2630: “Cryptographic Message Syntax”. TS 101 733 satisfies the requirements of 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 an 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 standard deals with the specification of new XML elements able to contain the signature policy information specified in TS 101 733.

There is a close relationship between the development of RFC 3126 and TS 101 733, which relate to ASN.1-based signatures, and between RFC 3275 and TS 101 933, which cover XML Digital Signature Syntax and Processing. It is hoped that, by staying abreast of these developments as they mature, this standard will remain consistent with the industry direction in this area. It should be noted that the SePS WSDL interface specification is a layer above the standards described in this Annex. Just as the SePS 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 E
(informative)

Relevant intellectual property rights (IPR)

E.1 Introduction

This informative annex lists all Intellectual Property Rights (trademarks, patents and patent applications), the use of which has been advised as **possibly** being implied by the application of this standard. It is stressed that the content of this annex is not exhaustive and is provided on a 'without prejudice' basis. That is:

— mention herein of a particular IPR indicates only that some party has expressed, to the Secretariat of CEN/TC 331 and/or the UPU Standards Board, the view that use of the standard might, in some circumstances, infringe the mentioned right. It should not be taken as in any way confirming the validity of such view and users of this standard should conduct their own searches to determine whether the mentioned IPR is in fact applicable to their specific case;

— the descriptive text associated with mention of a particular IPR is intended to provide only a general indication of the field of application of the right concerned. It should not be taken as implying that the right or its application is limited, in scope, to the description given;

— the absence of reference to any IPR is indicative only of the fact that the Secretariats of CEN/TC 331 and/or the UPU Standards Board had, up to the time of publication of this standard, received no suggestion that the said IPR could potentially be infringed by the use of this standard. Neither the UPU nor CEN has conducted any patent, trademark or other searches relevant to the subject matter of this standard and cannot accept any responsibility in case of infringement, on the part of users of this document, of any third party intellectual property rights.

Users of the standard are encouraged to conduct any necessary searches and to ensure that any pertinent IPR is either in the public domain; is licensed from the holder(s) or is avoided.

E.2 USPS Patents

The United States Postal Service claims the following issued and pending patents:

Filing Country	Patent or Application Number	Priority Date	Grant Date	Title
U.S.A.	US6917948 B2	20000908	20050712	Systems and methods for providing electronic archiving
U.S.A.	US7121455 B2	20010412	20061017	Systems and methods for electronic postmarking including ancillary data
U.S.A.	US7266696 B2	20001215	20070904	Electronic postmarking without directly utilizing an electronic postmark server
U.S.A.	AU7745000 A CA2386484 A1 CN1451213 EP1219063 A2 JP2003510962T T NZ518393 A US19990157168P WO0124437 A3	20010430 20010425 20001002 20020730 20030318 20050225 19990930 20001002	20030918	Systems and method for authenticating an electronic message
U.S.A.	WO02084945 US20040133524	20021024 20040708		Systems and methods for electronic postmarking of data including location data
U.S.A.	US20030177357 WO0217553	20000818 20020228		Apparatus and methods for the secure transfer of electronic data

In a letter dated 31 March 2006, the USPS has indicated that patent US6917948 concerns electronic archiving and may be relevant to performing certain aspects of the standard. The same letter indicates that the then patent applications, some of which have since been granted, may also be relevant to implementing certain aspects of the standard but that, as with all patent applications, the scope of the claimed inventions may change during examination of the applications. Thus, until the applications mature (if at all) to patents, it is not possible to determine their scope or their relative importance (or lack of importance) to the standard.

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.

[1] ISO 3166-1 list of country codes, http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html#af

Internet Engineering Task Force Public Key Infrastructure X.509 working group (IETF PKIX) documents

NOTE 1 Available at www.ietf.org.

[2] IETF Lightweight Directory Access Protocol (LDAP)

NOTE 2 This standard is adopted for data dissemination.

[3] RFC 2459 – Internet X.509 Public Key Infrastructure: Certificate and CRL Profile (January 1999), R. Housley, W. Ford, W. Polk, D. Solo

NOTE 3 The Certificate Revocation List profile used in this standard is the CRL profile defined in RFC 2459.

[4] RFC 2510 – Internet X.509 Public Key Infrastructure: Certificate Management Protocols (March 1999), C. Adams, S. Farrel

[5] RFC 2511 – Internet X.509: Certificate Request Message Format (March 1999), M. Myers, C. Adams, D. Solo, D. Kemp

[6] RFC 2527 – Internet X.509 Public Key Infrastructure: Certificate Policy and Certification Practices Framework (March 1999), S. Chockani, W. Ford

[7] 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

NOTE 4 Defines the ASN.1 layout for all relevant PKCS objects utilised by the SePS.

[8] RFC 2630 [2)] – Cryptographic Message Syntax (June 1999), R. Housley

[9] RFC 3280 – Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (supersedes RFC 2459)

European Telecommunications Standards Institute (ETSI) Standards

NOTE 5 ETSI standards are available at www.etsi.org.

[10] 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 SePS interface specification.

[11] 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 SePS 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 SePS 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.

This page deliberately left blank