|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **itu-old** | INTERNATIONAL TELECOMMUNICATION UNION | | | | COM 17 – C 268 – E |
| **TELECOMMUNICATION STANDARDIZATION SECTOR**  STUDY PERIOD 2013-2016 | | | **September 2014** | |
| **English only**  **Original: English** | |
| **Question(s):** | | 11/17 |  | | |
| **STUDY GROUP 17 – CONTRIBUTION 268** | | | | | |
| **Source:** | | Denmark | | | |
| **Title:** | | Proposal for including whitelist support in Rec. ITU-T X.509 | ISO/IEC 9594-8. | | | |

Justification

In some environment, such as smart grid, some entities only communicates with a limited set of other entities. Such environments are often constraint with respect to storage capacity, processing speed and bandwidth. Efficient public-key certificate handling is then essential. It is therefore proposed to include a whitelisting capability in Rec. ITU-T X.509 | ISO/IEC 9594-8 for efficient and speedy public-key certificate validation and for restricting communication to only selected parties.

IEC TC57 WG15 has identified a requirement for whitelisting in their work on IEC 62351-9, *Key Management*.

Whitelists are only relevant for end entities, i.e., entities to which end-entity public-key certificates have been issues. An end entity cannot issue public-key certificates to other entities.



Figure A – Certificate management

Figure A illustrates the basics of the proposed approach. End entity 1 (EE1) has communications requirements with end entities EE2, EE3, EE4 and EEa. A trusted external entity, a *delegator*, maintains information about the public-key certificate status for one or more end entities. This is illustrated in Figure A, where the delegator maintains public-key certificate information about EE1 and its potential communication partners including what is necessary for the delegator to maintain status information and perform validation on behalf of EE1. This includes maintaining certification path status information.

The issue for communication with EEa is a little more complicated, as it belongs to a PKI domain different from that of EE1. There are two options for the situation shown in Figure A:

a) The delegator, in addition to holding trust anchor information for trust anchor 1, also holds trust anchor information about trust anchor 2 and information about the relevant CAs subordinate to trust anchor 2, i.e., it holds information about CAb in figure A.

b) The CA2 and CAa have established cross certification meaning that it is possible to establish a certification path from trust anchor 1 to any entity subordinate to CAa.

The certification paths the delegator has to maintain on behalf of EE1 are as follows:

The certification path for PDUs received from EE2 is quite simple as EE1 and EE2 share the same issuing CA:

CA1 >> EE2 (the trust anchor is not part of the certification path).

The certification path for PDUs received from EE3 or EE4 is similarly quite simple as they share the same trust anchor:

CA2 >> EE3 and CA2 >> EE4.

The certification path for PDUs received from EEa is a little more complicated. Two cases are considered. The first case is where the EE1 has trust anchor information from trust anchor 2:

CAa >> EEa

If, at the other hand, EE1 has no trust anchor information from trust anchor 2, but there is cross certification between CA2 and CAa, then the certification path is:

CA2 >> CAa >> EEa

If an incident happens that invalidates a certification path, EE1 has to be informed about the change in the status.

Based on the information the delegator holds, it constructs a whitelist to be forwarded to EE1. This whitelist contains sufficient information to allow EE1 to validate requests/responses from other end entities without having to consult a third party.

The delegator maintains the whitelist at EE1 by either a replace request or an update request. In particular, the EE1 must be kept updated on any change in the status information. It is proposed that the delegator maintains status information by a subscription service.

The communications between EE1 and the delegator need to be secure, i.e., it is necessary to ensure integrity, confidentiality and authentication. It is proposed to use Cryptographic Message Syntax (CMS) for enveloping the communications.

A delegator may be closely related to the end entities it is supporting, e.g., they may be run may the same organisation, meaning that the end entities can consider the delegator as a trust anchor. This allows for an efficient validation of the whitelist.

A delegator may acts as delegator for multiple end entities providing similar information as illustrated in figure A for several end entities.

As mentioned before, the EE1 holds a whitelist signed by the delegator. I addition, it holds its own public-key certificate and its private key. In this simple example, it also holds delegator trust anchor information.



Figure B – Delegator communications partners

Figure B illustrates the communication partners with which the delegator is communicating. It provides status update information to supporting end entities and it receives status information from CAs responsible public-key certificates that affects the certifications paths for the supporting end entities.



Figure C – Whitelist management

Figure C illustrates the different type of communications between the delegator and the end entity, for which the delegator maintains a whitelist.



Figure D – Delegator management

Figure D illustrates the different type of communications between the delegator and one of the CAs from which it can subscribe on public-key certificate status information.

Proposal for addition to Rec. ITU-T X.501 | ISO/IEC 9594-2

*Add new object identifiers to the UsefulDefinitions module to allow for CMS support within Rec. ITU-T X.509 | ISO/IEC 9594-8:*

*Add the following new attribute type for allocation of object identifiers for CMS content types:*

cmsContentType ID ::= {ds 41}

*Add a new object identifier for CMS a module holding CMS content specifications*

cmsContentSpecifications ID ::= {module cmsContentSpecifications(40) 8}

*Add a new synonym:*

id-cmsct ID ::= cmsContentType

Proposal for addition to Rec. ITU-T X.509 | ISO/IEC 9594-8

*Add the following abbreviation to clause 4:*

CASP Certification Authority Subscription Protocol

WLMP WhiteList Management Protocol

*Add a new clause 6.3 and renumber subsequent clauses as appropriate:*

## 6.3 Fingerprints

When comparing for equality two values of a structured data type or two large values of a simple data type use of fingerprints could ease such a comparison. A fingerprint of a value is short representation of the value by some algorithm, e.g., by providing a hash of the value.

The following data type may be used to define the data type resulting from a particular algorithm, e.g. a hash algorithm, when applied to values of a specified data type:

Fingerprint {ToBeFingerprinted} ::= SEQUENCE {

algorithmIdentifier AlgorithmIdentifier{{SupportedAlgorithms}},

fingerprint BIT STRING,

... }

For two otherwise equal values, the fingerprint comparison will fail if the two values are not encoding in the same way, i.e., before creating the fingerprint, the two values shall be encoded using the same canonical encoding rule, e.g., DER.

*Add a new clause 11 and renumber subsequent clauses:*

# 11 Public-key certificate whitelisting

## 11.1 Whitelist concept

In some environments, end entities may only communicate with a few other end entities and will not accept PDUs from any other entity. Validation may be optimized in such environment by use of whitelists. A whitelist is a list providing information about potential communications partners for a particular end entity. If a PDU is received from an entity not represented in the whitelist, the PDU shall be discarded. The whitelist is created, maintained and signed by an entity called the delegator, which is an entity to which an end entity has delegated part of the maintenance and validation tasks. For efficiency, the delegator could be a trust anchor for the end entity.

Two scenarios are recognized here:

a) A whitelist is placed in an end entity with no or minor constraints allowing the end entity to perform much of the validation on its own. In this case, the whitelist is mainly used to restrict the communications to selected entities.

b) A whitelist is placed in an entity that is constraints with respect to processing power, storage and/or response time to a degree that it cannot afford to go to a third party when doing validation, meaning that the end entity locally needs to have locally sufficient updated information available to do the validation.

## 11.2 The delegator

The behaviour of the delegator is dependent on whether it maintains whitelists for end entities in a non-constrained environment or for end entities in a constrained environment.

If the whitelist is to be placed in a non-constraint end entity, the whitelist shall only contain rather stable information not affected by state or change of associated public-key certificates.

If the whitelist is to be placed in a constraint end entity, the delegator shall maintain a complete certification paths required for each of the peer entities with which the end entity communicates. The delegator shall continuously ensure that the certification paths can validate positively. This included checking for restrictions and expired or revoked public-key certificates. When a certification path cannot longer validate positively, the corresponding whitelist item shall be updated within the affected end entity.

## 11.3 Whitelist syntax

The CertWhitelist ASN.1 data type specifies a whitelist.

CertWhitelist ::= SIGNED {TBSCertWhitelist}

TBSCertWhitelist ::= SEQUENCE {

version [0] Version DEFAULT v1,

serialNumber WhitelistSerialNumber OPTIONAL,

signature AlgorithmIdentifier {{SupportedAlgorithms}},

issuer Name,

constraint BOOLEAN DEFAULT TRUE,

certInfo SEQUENCE OF SEQUENCE {

certIdentifier PKCertIdentifier,

certStatus CertStatus OPTIONAL,

...,

...,

wlEntryExtensions [1] Extensions OPTIONAL },

...,

...,

wlExtensions Extensions OPTIONAL }

(CONSTRAINED BY { -- shall be DER encoded -- } )

WhitelistSerialNumber ::= INTEGER (0..MAX)

PKCertIdentifier ::= CHOICE {

issuerSerialNumber IssuerSerialNumber,

fingerprintPKC [0] FINGERPRINT {TBSCertificate},

fingerprintPK [1] FINGERPRINT {PublicKey},

... }

IssuerSerialNumber ::= SEQUENCE {

issuer Name,

serialNumber CertificateSerialNumber,

... }

CertStatus ::= ENUMERATED {

good (0),

revoked (1),

on-hold (2),

expired (3),

... }

The whitelist components are specified in the following.

The version component shall hold the version of the whitelist. This component shall either be absent or have the value v1.

The serialNumber component, when present, shall contain a serial number that is unique within the end entity where the whitelist is placed. This component shall be present if the delegator places more than one whitelist in a particular end entity. Otherwise, it is optional.

The signature component shall contain the algorithm identifier for the signature algorithm used by the delegator when signing the whitelist. This component shall be the same value as used in the algorithmIdentifier component of the SIGNATURE data type when signing the whitelist.

NOTE – By including this component, the signature algorithm is protected by the signature on the whitelist.

The issuer component shall be the distinguished name of the delegator issuing the whitelist.

The constraint component shall take the value FALSE if the end entity in not resource constraint and therefore has the resources to perform normal validation, e.g., to make use of the OCSP service. Otherwise, this component shall take the value TRUE or be absent.

The certInfo component shall hold an element that for each of the end-entity public-key certificates represented by the whitelist. Each such element shall hold a value with the following components:

a) The certIdentifier component shall hold an identifier of a public-key certificate represented by the element and it has the following alternatives:

– issuerSerialnumber, which is a sequence of the distinguished name of the public-key certificate issuer and the public-key certificate serial number;

– fingerprintPKC, which is a fingerprint of the unsigned public-key certificate;

– fingerprintPK, which is a fingerprint of the public key within the public-key certificate.

b) The certStatus component shall be absent if the constraint component has the value FALSE. Otherwise, it shall be present and shall then hold the status of the public-key certificate represented by this element:

– the good value signals that the represented public-key certificate can be trusted;

– the revoked value signals that the represented public-key certificate has been revoke and cannot longer be trusted;

– the on-hold value signals that the represented public-key certificate has been put on hold status and should not be trusted for the time being;

– the expired value signals that the represented public-key certificate has expired and cannot longer be trusted.

c) The wlEntryExtensions component, if present, shall contain one or more whitelist entry extensions.

NOTE – Additional restriction may reflected in an entry extension.

The wlExtensions component, if present, shall contain one or more whitelist extensions.

*Add a new SECTION 4.*

SECTION 4 – COMMUNICATIONS CAPABILITIES

# 20 Use of cryptographic message syntax (CMS)

Cryptographic message syntax (CMS) is defined in Rec. ITU-T X.CMS. It defines communication capabilities that allow for data integrity, confidentiality and authentication. CMS may be used for maintaining PKI and PMI related information.

The CMS defines different content types to be used for different purposes.

CONTENT-TYPE ::= TYPE-IDENTIFIER

The CONTENT-TYPE information object class is equivalent to the ASN.1 built information object class TYPE-IDENTIFIER. The CONTENT-TYPE information object is used to bind the content type to the abstract syntax of the content.

This Specification defines specific content types to be enveloped by the signed content type, the signed and encrypted content type or to be transmitted as non-enveloped data.

# 21 Whitelist management and certification authority subscription protocols

The whitelist management protocol (WLMP) is used between a delegator and an end entity for whitelist management. The CA subscription protocol (CASP) is used between a delegator and a CA to which the delegator subscribes to public-key certificate status.

## 21.1 Use of cryptographic message syntax signedData content type

The ct-signedData content type is defined in Rec. ITU-T X.CMS. This Specification puts some restriction on its use. This is formally specified by the use of an equivalent wlSignedData content type identified by the same object identifier as the ct-signedData content type. An implementation of the wlSignedData content type is conformant with the ct‑signedData content type.

The whitelist signed data content is defined as:

wlSignedData CONTENT-TYPE ::= {

WLSignedData

IDENTIFIED BY id-signedData }

WLSignedData ::= SEQUENCE {

version CMSVersion (v3),

digestAlgorithms SET (SIZE (1)) OF AlgorithmIdentifier {{WL-Hash-Algorithms}},

encapContentInfo EncapsulatedContentInfo,

certificates [0] IMPLICIT SET (SIZE (1..MAX)) OF Certificate OPTIONAL,

--crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,

signerInfos SignerInfos,

... }

EncapsulatedContentInfo ::= SEQUENCE {

eContentType CONTENT-TYPE.&id({WLContentSet}),

eContent [0] EXPLICIT OCTET STRING

(CONTAINING CONTENT-TYPE.&Type({WLContentSet}{@eContentType}))}

SignerInfos ::= SET (SIZE (1)) OF SignerInfo

SignerInfo ::= SEQUENCE {

version CMSVersion,

sid SignerIdentifier,

digestAlgorithm AlgorithmIdentifier {{WL-Hash-Algorithms}},

signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,

signatureAlgorithm AlgorithmIdentifier {{WL-Signature-Algorithms}},

signature SignatureValue,

unsignedAttrs [1] IMPLICIT Attributes{{UnsignedAttributes}} OPTIONAL }

SignerIdentifier ::= CHOICE {

--issuerAndSerialNumber IssuerAndSerialNumber,

subjectKeyIdentifier [0] SubjectKeyIdentifier,

--certHash [1] CertHash,

...}

WL-Hash-Algorithms ALGORITHM ::= {...}

WL-Signature-Algorithms ALGORITHM ::= {...}

The WLSignedData data type has the components specified in the following.

The version component shall take the value v3.

The digestAlgorithms component shall consist of a single element specifying a hashing algorithm from the set of applicable hashing algorithms.

The encapContentInfo component shall specify the set of content types applicable for the whitelist support.

The certificates component, when present, shall specify the set of public-key certificates that makes up the certification path to be used for signature verification.

The crls component shall be absent.

The signerInfos shall consists of a single element with the following components:

a) The version component shall take the value v3. ??

b) The sid component shall take the subjectKeyIdentifier alternative.

c) The digestAlgorithm component shall be a hash algorithm of the repertoire specified by the WL-Hash-Algorithms set.

d) The signedAttrs component - necessary?

e) The signatureAlgorithm component shall be a signature algorithm of the repertoire specified by the WL-Signature-Algorithms set.

f) The signature component

g) The unsignedAttrs component - necessary?

## 21.2 Use of cryptographic message syntax signcryptedData content type

*To be completed when Rec. ITU-T X.CMS has been further developed.*

## 21.3 Content types specific for whitelist support

The following CMS content types are defined for the WLMP and the CASP:

WLContentSet CONTENT-TYPE ::= {

addWhitelistReq |

addWhitelistRsp |

replaceWhitelistReq |

replaceWhitelistRsp |

updateWhitelistReq |

updateWhitelistRsp |

deleteWhitelistReq |

deleteWhitelistReq |

rejectWhitelist |

certSubscribeReq |

certSubscribeRsp |

certUnsubscribeReq |

certUnsubscribeRsp |

certReplaceReq |

certReplaceRsp |

certUpdateReq |

certUpdateRsp |

rejectCAsubscribe,

... }

## 21.4 Checking of received content

When a message is received, the recipient shall perform a number of validation steps. If the validation fails at any step, no further validation is necessary and the recipient shall return an appropriate error code. Error codes for the WLMP and CASP are specifies in clauses 21.5.8 and 21.6.8.

A number of checks are common across different content types. Such common checks are specified in the following. The receiver of a content shall check:

a) whether the content type is a supported one and if not, return an unknownContentType error code;

b) whether the version component of the content is supported and if not, return an unsupportedWLMPversion or unsupportedCASPversion error code, as appropriate;

c) whether the content is present and if not, return a missingContent error code;

d) whether all mandatory content components are present and if not, return a missingContentComponent error code;

e) whether unexpected components are included in the content and if so, return an invalidContentComponent error code;

f) whether the sequence component hold a valid sequence number and if not, return a sequenceError error code as specified in clause 21.5.8 for the WLMP and in clause 21.6.8 for CASP.

## 21.5 Whitelist management protocol

### 21.5.1 Whitelist management introduction

The whitelist management is concerned with how the delegator maintains whitelist information within the end entities it supports. It encompasses a set of CMS exchanges as detailed in the following.

### 21.5.2 Whitelist common components

Some components are common across different content types. The WLMPcommonComponents data type comprises these components.

WLMPcommonComponents ::= SEQUENCE {

version WLMPversion DEFAULT v1,

sequence WLMPsequence,

... }

WLMPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

WLMPsequence ::= INTEGER (1..MAX)

The WLMPcommonComponents data type has the following components.

The version component shall hold the version of the WLMP. The current version is version v1.

The sequence component shall hold a sequence number of a message being sent. The sequence number is used:

a) to allow detection replay of messages caused by an error or by a hostile attacker;

b) to pair requests and responses;

c) to detect missing messages.

*Editor's note – Maybe we need more here, time stamp, nonce, etc.*

### 21.5.3 Add whitelist

addWhitelistReq CONTENT-TYPE ::= {

AddWhitelistReq

IDENTIFIED BY id-addWhitelistReq }

The delegator uses the addWhitelistReq content type to initiate the addition of a whitelist to an end entity.

AddWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

certlist CertWhitelist,

... }

The AddWhitelistReq data type specifies the actual content and has the following components:

a) The components of the WLMPcommonComponents data type is specified in clause 21.5.2. The sequence component shall take the value 1.

b) The certList component shall hold the whitelist to be added to the end entity.

The receiving end entity shall check the validity of the request:

a) by checking as specified in clause 21.4;

b) by checking the validity of the signature on the received whitelist and if invalid, return an invalidSignature error code;

c) if the serialNumber component is present in the whitelist, then check whether a whitelist with the same WhitelistSequenceNumber value already exists and if so, return a duplicateWL error code;

d) if the serialNumber component is absent in the whitelist, then check whether a whitelist with absent serialNumber component already exists and if so, return a duplicateWL error code;

e) by checking whether all whitelist mandatory components are present and if not, return a missingWLcomponent error code;

f) by checking whether the version component on the whitelist specifies a supported version and if not, return a invalidWLversion error code;

g) by checking whether the constraint component on the received whitelist indicates a supported constraint mode and if not, return a constraintError error code;

h) by checking whether the certStatus component specifies an unknown status code and if so, return an unknownCertStatus error code;

i) by checking whether the whitelist contains an unsupported critical extension and if so, return an unsupportedCriticalExtenssion error code;

j) by checking whether the maximum number of whitelists has been exceeded by the new whitelist and if so, return a maxWLsExcited error code.

NOTE – Maximum limit might be just a single whitelist.

addWhitelistRsp CONTENT-TYPE ::= {

AddWhitelistRsp

IDENTIFIED BY id-addWhitelistRsp }

The end entity uses the addWhitelistRsp content type to report the outcome of an add whitelist request.

AddWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] AddWhitelistOK,

failure [1] AddWhitelistErr,

... },

... }

AddWhitelistOK ::= SEQUENCE {

ok NULL,

... }

AddWhitelistErr ::= SEQUENCE {

notOK WLMP-error,

... }

The AddWhitelistRsp data type specifies the actual content and has the following components:

The components of the WLMPcommonComponents data type is specified in clause 21.5.2.

The result component has the following alternatives:

a) The success alternative shall be taken if the addition of a whitelist was performed successfully.

b) The failure alternative shall be taken if the addition of a whitelist failed. The WLMP-error data type is specified in clause 21.5.8.

### 21.5.4 Replace whitelist

replaceWhitelistReq CONTENT-TYPE ::= {

ReplaceWhitelistReq

IDENTIFIED BY id-replaceWhitelistReq }

The delegator uses the replaceWhitelistReq content type to initiate the replacement of a whitelist at an end entity. It shall be used when one or more public-key certificates represented by the whitelist has been replaced or when the delegator key information has changes.

ReplaceWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

old WhitelistSerialNumber OPTIONAL,

new CertWhitelist,

... }

The ReplaceWhitelistReq data type specifies the actual content and has the following components:

a) the components of WLMPcommonComponents data type as specified in clause 21.5.2;

b) the old component, when present, shall hold the serial number of the old whitelist; and

c) the new component shall hold the replacement whitelist.

The end entity shall verify the validity of the request by checking:

a) as specified in 21.5.3 items a) to i);

b) if the old component was present in the request, then check whether the WhitelistSerialNumber value specified in that component matches the WhitelistSerialNumber of a local whitelist and if not, return an unknownWL error code.

c) if the old component was absent in the request, then check whether there locally is just a single whitelist and that whitelist is without the serialNumber component and if not, return an unknownWL error code.

replaceWhitelistRsp CONTENT-TYPE ::= {

ReplaceWhitelistRsp

IDENTIFIED BY id-replaceWhitelistRsp }

The end entity uses the replaceWhitelistRsp content type to report the outcome of a whitelist replace request.

ReplaceWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] RepWhitelistOK,

failure [1] RepWhitelistErr,

... },

... }

RepWhitelistOK ::= SEQUENCE {

ok NULL,

... }

RepWhitelistErr ::= SEQUENCE {

notOK WLMP-error,

... }

The ReplaceWhitelistRsp data type specifies the actual content and has the following components:

The components of the WLMPcommonComponents data type as specified in clause 21.5.2.

The result component has the following alternatives:

a) The success alternative shall be taken if the replacement of a whitelist was performed successfully.

b) The failure alternative shall be taken if the replacement of a whitelist failed. The WLMP-error data type is specified in clause 21.5.8.

### 21.5.5 Update whitelist

updateWhitelistReq CONTENT-TYPE ::= {

UpdateWhitelistReq

IDENTIFIED BY id-updateWhitelistReq }

The delegator uses the updateWhitelistReq content type to initiate updates of a whitelist at an end entity. This request content type is only relevant if the whitelist in question has the constraint component set to TRUE or absent. It shall be used when the status of one or more public-key certificates represented by the whitelist has changed.

UpdateWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

wl-Id WhitelistSerialNumber OPTIONAL,

status SEQUENCE (SIZE (1..MAX)) OF WhitelistStatus,

signature WLsignature,

... }

WhitelistStatus ::= SEQUENCE {

subjectId PKCertIdentifier,

update CertStatus,

... }

WLsignature ::= ENCRYPTED-HASH {TBSCertWhitelist}

The UpdateWhiteListReq data type specifies the actual content and has the following components:

a) The components of WLMPcommonComponents data type as specified in clause 21.5.2.

b) The wl-Id component shall identify the whitelist to be updated.

c) The status component shall hold a list of status changes and each element has he following subcomponents:

– The subjectId subcomponent shall identify the particular public-key certificate for which the status has changed.

– The update subcomponent shall hold the updated status of the public-key certificate.

d) The signature component shall hold an updated signature of the whitelist reflecting the whitelist after it has been updated.

The end entity shall verify the validity of the request by checking

a) whether the end entity is resource constrained and if not,, return a notRelevantContent error code;

b) as specified in clause 21.4;

c) if the wl-id component was present in the request, then check whether the WhitelistSerialNumber value specified in that component matches the WhitelistSerialNumber of a local whitelist and if not, return an unknownWL error code.

d) if the wl-id component was absent in the request, then check whether there locally is just a single whitelist without the serialNumber component and if not, return an unknownWL error code.

e) each element of the status component as to

– whether the subjectId subcomponent matches the identity of public-key certificate represented by the identified whitelist and if not, return a unknownCert error code;

– whether the update subcomponent specifies an unknown status code and if so, return a unknownCertStatus error code;

f) whether the signature component is valid for the updated whitelist and if not, return an invalidSignature error code.

updateWhitelistRsp CONTENT-TYPE ::= {

UpdateWhitelistRsp

IDENTIFIED BY id-updateWhitelistRsp }

The end entity uses the updateWhitelistRsp content type to report the outcome of an update request.

UpdateWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] UpdWhitelistOK,

failure [1] UpdWhitelistErr,

... },

... }

UpdWhitelistOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

result NULL,

... },

not-ok [1] SEQUENCE {

status WLMP-CertStatusCode,

... },

... }

WLMP-CertStatusCode ::= ENUMERATED {

noReason (1),

unknownCert (2),

unknownCertStatus (3),

... }

UpdWhitelistErr ::= SEQUENCE {

notOK WLMP-error,

... }

The UpdateWhitelistRsp data type specifies the actual content and has the following components:

The components of the WLMPcommonComponents data type as specified in clause 21.5.2.

The result component has the following alternatives:

a) The success alternative shall be taken if at least one update of the whitelist was performed successfully. It shall then hold a value of the UpdWhitelistOK data type.

b) The failure alternative shall be taken if the update of a whitelist failed to a degree where no result could be returned. It shall then hold a value of UpdWhitelistErr data type. The WLMP-error data type is specified in clause 21.5.8.

The UpdWhitelistOK shall include an element for each public-key certificate specified in the request in the same order. Each of the element has two alternatives:

a) The ok alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

– the result component indicates that element was successfully updated.

– the status component shall hold the status of the public-key certificate as defined in clause 11.3.

b) The not-ok alternative shall be taken when the corresponding element was not updated. It shall take one of the following values:

– the no-reason status code shall be returned when no other status code is applicable;

– the unknownCert status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA;

– the unknownCertStatus status code shall be selected when an unknown status code was received.

### 21.5.6 Delete whitelist

deleteWhitelistReq CONTENT-TYPE ::= {

DeleteWhitelistReq

IDENTIFIED BY id-deleteWhitelistReq }

The delegator uses the deleteWhitelistReq content type to initiate deletion of a whitelist at an end entity.

DeleteWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

wl-Id WhitelistSerialNumber OPTIONAL,

... }

The DeleteWhitelistReq data type specifies the actual content and has the following component:

a) The components of WLMPcommonComponents data type as specified in clause 21.5.2.

b) The wl-Id component, when present, shall identify the whitelist to be deleted.

The end entity shall verify the validity of the request by checking

a) as specified in clause 21.4;

b) if the wl-id component was present in the request, then check whether the WhitelistSerialNumber value specified in that component matches the WhitelistSerialNumber of a local whitelist and if not, return an unknownWL error code.

c) if the wl-id component was absent in the request, then check whether there locally is just a single whitelist and that whitelist is without the serialNumber component and if not, return an unknownWL error code.

deleteWhitelistRsp CONTENT-TYPE ::= {

DeleteWhitelistRsp

IDENTIFIED BY id-deleteWhitelistRsp }

The end entity uses the deleteWhitelistRsp content type to report the outcome of a delete request.

DeleteWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] DelWhitelistOK,

failure [1] DelWhitelistErr,

... },

... }

DelWhitelistOK ::= SEQUENCE {

ok NULL,

... }

DelWhitelistErr ::= SEQUENCE {

notOK WLMP-error,

... }

The DeleteWhitelistRsp data type specifies the actual content. It has the following components:

The components of the WLMPcommonComponents data type as specified in clause 21.5.2.

The result component has the following alternatives:

a) The success alternative shall be taken if the deletion of a whitelist was performed successfully.

b) The failure alternative shall be taken if the deletion of a whitelist failed. The WLMP-error data type is specified in clause 21.5.8.

### 21.5.7 Whitelist reject

rejectWhitelist ::= CONTENT-TYPE ::= {

RejectWhitelist

IDENTIFIED BY id-rejectWhitelist }

The rejectWhitelist content type is used by the delegator to report problems with a response from the end entity.

RejectWhitelist ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

reason WLMP-error,

... }

The RejectWhitelist data type specifies the actual content and has the following component:

The sequence component of the WLMPcommonComponents data type shall take the same value as in the response on which it is reporting.

The WLMP-error is specified in clause 21.5.8.

The delegator shall verify the validity of a received response by checking

a) as specified in clause 21.4.

### 21.5.8 Whitelist error codes

A value of the WLMP-error data type is used by the end entity to report an error when processing a request from the delegator. It is also used by a delegator to reject a faulty response from an end entity.

WLMP-error ::= ENUMERATED {

noReason (0),

unknownContentType (1),

unsupportedWLMPversion (2),

missingContent (3),

missingContentComponent (4),

invalidContentComponent (5),

sequenceError (6),

notRelevantContent (7).

invalidSignature (8),

duplicateWL (9),

missingWLcomponent (10),

invalidWLversion (11),

constraintError (12),

unknownCertStatus (13),

unsupportedCriticalExtenssion (14),

maxWLsExceeded (15),

unknownCert (16),

unknownWL (17),

... }

a) the noReason value shall be selected when no other error code is applicable;

b) the unknownContentType value shall be selected if the content type is not known by the receiver;

c) the unsupportedWLMPversion value shall be selected if a request or response content specified a WLMP version not supported;

d) the unsupportedContentVersion value shall be selected when a request or response includes an unsupported content type;

e) the missingContent value shall be selected when the request or response did not include a content;

f) the missingContentComponent value shall be selected when a request or response did not includes a mandatory component;

g) the invalidContentComponent value shall be selected when an unexpected component was included in a request or response;

h) the sequenceError value shall be selected by when:

– an end entity receives a request content of the addWhitelistReq content type that did not have the sequence component set to 1;

– an end entity receives a request content not of the addWhitelist content type that did not have the sequence component set to one more than for the previous request; or

– a delegator receives a response content with a sequence component value different from the one in the corresponding request content;

i) the notRelevantContent value shall be selected if a content type is not relevant for a non-constraint end entity;

j) the invalidSignature value shall be selected when the signature on a whitelist is invalid;

k) the duplicateWL value shall be selected when delegator attempts to add en already existing whitelist to an end entity;

l) the missingWLcomponent value shall be selected when a received whitelist is missing a mandatory component;

m) the invalidWLversion value shall be selected when an unsupported whitelist version is received;

n) the constraintError value shall be selected when a received whitelist has an invalid constraint component;

o) the unknownCertStatus value shall be selected when a received whitelist or a whitelist update that contained an unknown public-key certificate status;

p) the unsupportedCriticalExtenssion value shall be selected when a reived whitelist contains an unsupported critical extension;

q) the maxWLsExceeded value shall be selected when the addition of a whitelist would bring the number of whitelist beyond a locally determined value;

r) the unknownCert value shall be selected when an unknown public-key certificate was referenced in an update request;

s) the unknownWL value shall be selected when an end entity receives a content including a value of the WhitelistIdentifier data type that did not match any local whitelist.

## 21.6 Certification authority subscription protocol

### 21.6.1 Certification authority subscription introduction

The certification authority subscription is concerned with how the delegator maintains whitelist status information by subscribing to the necessary information from relevant CAs. It is only relevant for delegators supporting whitelists for constraint end entities.

Before subscribing to maintenance information, the delegator needs to know the exact certification configuration for the end entities it supports. The following information is necessary to establish:

a) The end-entity public-key certificates for the end entities for which whitelist support is to be provided.

b) For each end entities from a), the end-entity public-key certificates for the end entities to which communications are possible.

c) The CA-certificates and trust anchor information necessary to establish any necessary certification path.

This Specification does not gives details on how a delegator obtains this information. It could be by local configuration or by abstract of a centralized database.

The CASP comprises a set of CMS exchange types as detailed in the following.

### 21.6.2 Certification authority subscription common components

Some components are common across different content types. The CASPcommonComponents data type comprises these components.

CASPcommonComponents ::= SEQUENCE {

version CASPversion DEFAULT v1,

sequence CASPsequence,

... }

CASPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

CASPsequence ::= INTEGER (1..MAX)

The CASPcommonComponents data type has the following components.

The version component shall hold the version of the CASP. The current version is version v1.

The sequence component shall hold a sequence number of a message being sent. The sequence number is used:

a) to allow detection replay of messages cause by an error or by a hostile attacker;

b) to pair requests and responses;

c) to detect missing messages.

### 21.6.3 Public-key certificate subscription

certSubscribeReq CONTENT-TYPE ::= {

CertSubscribeReq

IDENTIFIED BY id-certSubscribeReq }

The delegator uses the certSubscribeReq content type to request a specific CA to supply status information about public-key certificates issued by this CA relevant for the whitelists supported by the delegator.

CertSubscribeReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE OF SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

... }

The CertSubscribeReq data type specifies the actual content and has the following components:

a) The components of the CASPcommonComponents data type as specified in clause 22.6.2.

b) The certs component shall identify a list of public-key certificates for which, the delegator requests information about status changes. It has the following subcomponents for each element:

– the subject subcomponent shall be the name of the entity for which the public-key certificate has been issued;

– the serialNumber subcomponent shall be the serial number for the public-key certificate in question.

The CA shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the certs component for validity, i.e., whether it identifies a public-key certificate issued by the CA. If not, an unknownCert status code shall be returned in the corresponding element of the response.

certSubscribeRsp CONTENT-TYPE ::= {

CertSubscribeRsp

IDENTIFIED BY id-certSubscribeRsp }

The CA shall use the certSubscribeRsp content type to report the outcome of the subscription request.

CertSubscribeRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertSubscribeOK,

failure [1] CertSubscribeErr,

... },

... }

CertSubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

cert Certificate,

status CertStatus,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CASP-CertStatusCode ::= ENUMERATED {

noReason (1),

unknownCert (2),

... }

CertSubscribeErr ::= SEQUENCE {

code CASP-error,

... }

The CertSubscribeRsp data type specifies the actual content and has the following components:

The components of the CASPcommonComponents data type as specified in clause 21.6.2.

The result component has the following two alternatives:

a) The success alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the CertSubscribeOK data type.

b) The failure alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the CertSubscribeErr data type. The CASP-error data type is specified in clause 21.6.8.

The CertSubScribeOK shall include an element for each public-key certificate specified in the request in the same order. Each of the element has two alternatives:

a) The ok alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

– the cert component shall hold the public-key certificate for the requested subject;

– the status component shall hold the status of the public-key certificate as defined in clause 11.3.

b) The not-ok alternative shall be taken when a corresponding public-key certificate was not identified:

– the no-reason status code shall be returned when no code is applicable;

– the unknownCert status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

### 21.6.4 Public-key certificate un-subscription

certUnsubscribeReq CONTENT-TYPE ::= {

CertUnsubscribeReq

IDENTIFIED BY {id-cmsct 10} }

The delegator uses the certUnsubscribeReq content type to request a specific CA to stop supplying status information about public-key certificates issued by that CA.

CertUnsubscribeReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE OF SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

... }

The CertUnsubscribeReq data type specifies the actual content and has the following components:

a) The components of CASPcommonComponents data type as specified in clause 22.6.2.

b) The certs component shall identify a list of public-key certificates for which, the delegator requests stop for information about status changes. It has the following subcomponents for each public-key certificate:

– The subject subcomponent shall be the name of the entity to which the public-key certificate has been issued.

– The serialNumber subcomponent shall be the serial number for the public-key certificate in question.

The CA shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the certs component for validity, i.e., whether it identifies a public-key certificate issued by the CA. If not, an unknownCert status code shall be returned in the corresponding element of the response.

certUnsubscribeRsp CONTENT-TYPE ::= {

CertUnsubscribeRsp

IDENTIFIED BY id-certUnsubscribeReq } }

The CA shall use the certUnsubscribeRsp content type to report the outcome of the un-subscription request.

certUnsubscribeRsp CONTENT-TYPE ::= {

CertUnsubscribeRsp

IDENTIFIED BY id-certUnsubscribeRsp }

CertUnsubscribeRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertUnsubscribeOK,

failure [1] CertUnsubscribeErr,

... },

... }

CertUnsubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CertUnsubscribeErr ::= SEQUENCE {

code CASP-error,

... }

The CertSubscribeRsp data type specifies the actual content and has the following components:

The components of the CASPcommonComponents data type as specified in clause 21.6.2.

The result component has following two alternatives:

The result component has the following two alternatives:

a) The success alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the CertUnsubscribeOK data type.

b) The failure alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the CertUnsubscribeErr data type. The CASP-error data type is specified in clause 21.6.8.

The CertUnsubScribeOK includes an element for each public-key certificate specified in the request in the same order. Each of the element has two alternatives:

a) The ok alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

– the subject component shall hold the name of the subject to which the public-key certificate had been issued;

– the serialNumber component shall hold the serial number for the public-key certificate.

b) The not-ok alternative shall be taken when a corresponding public-key certificate was not identified.

– the no-reason status code shall be returned when no other status code is applicable;

– the unknownCert status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

The error alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. The CASP-error data type is specified in clause 21.6.8.

### 21.6.5 Public-key certificate replacements

certReplaceReq CONTENT-TYPE ::= {

CertReplaceReq

IDENTIFIED BY id-certReplaceReq }

The CA shall use the certReplacementReq content type to submit replaced public-key certificates to the delegator.

CertReplaceReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE OF SEQUENCE {

old CertificateSerialNumber,

new Certificate,

... },

... }

The CertReplacementReq data type specifies the actual content and has the following components:

a) The components of the CASPcommonComponents data type as specified in clause 21.6.2.

b) The certs component shall identify a list of public-key certificate replacements. It has the following subcomponents for each public-key certificate:

– The old subcomponent shall hold the identification of the public-key certificate to be replaced.

– The new subcomponent shall hold the replacement public-key certificate.

The delegator shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the certs component for validity:

– whether the old subcomponent identifies a public-key certificate at the delegator and if not, an unknownCert status code shall be returned in the corresponding element of the response;

b) each element of the certs component for validity, i.e., whether it identifies a public-key certificate issued by the CA. If not, an unknownCert status code shall be returned in the corresponding element of the response.

certReplaceRsp CONTENT-TYPE ::= {

CertReplaceRsp

IDENTIFIED BY id-certReplaceRsp }

The delegator shall use the certReplacementRsp content type to report the outcome of the subscription request.

CertReplaceRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertReplaceOK,

failure [1] CertReplaceErr,

... },

... }

CertReplaceOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

issuer Name,

serialNumber CertificateSerialNumber,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CertReplaceErr ::= SEQUENCE {

code CASP-error,

... }

The CertReplaceRsp data type specifies the actual content and has the following components:

The components of the CASPcommonComponents data type as specified in clause 21.6.2.

The result component has following two alternatives:

a) The success alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the CertReplaceOK data type.

b) The failure alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the CertReplaceErr data type. The CASP-error data type is specified in clause 21.6.8.

The CertReplace data type includes an element for each public-key certificate specified in the request in the same order. Each of the element has two alternatives:

a) The ok alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

– the subject component shall hold the name of the subject to which the public-key certificate had been issued;

– the serialNumber component shall hold the serial number for the public-key certificate.

b) The not-ok alternative shall be taken when a corresponding public-key certificate was not identified.

– the no-reason status code shall be returned when no code is applicable;

– the unknownCert status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

### 21.6.6 Public-key certificate updates

certUpdateReq CONTENT-TYPE ::= {

CertUpdateReq

IDENTIFIED BY id-certUpdateReq }

The CA shall use the certUpdateReq content type to submit to the delegator updated status information on public-key certificates.

CertUpdateReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

certStatus CertStatus,

... },

... }

The CertUpdateReq data type specifies the actual content and has the following components:

a) The components of the CASPcommonComponents data type as specified in clause 21.6.2.

b) The certs component shall identify a list of updates to public-key certificate. It has the following subcomponents for each element:

– The subject subcomponent shall hold the identification of the public-key certificate to be replaced.

– The serialNumber subcomponent shall identify the public-key certificate or which new status information is available.

– The certStatus shall hold the updated status information for the public-key certificate in question.

The delegator shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the certs component for validity by checking–:

– whether the subject subcomponent identifies a new entity and if not, return an unknownSubject error code;

– whether the serialNumber subcomponent identifies a known public-key certificate and if not, return an unknownCert error code;

– whether certStatus subcomponent has valid value and if not, return an unknownCertStatus error code.

certUpdateRsp CONTENT-TYPE ::= {

CertUpdateRsp

IDENTIFIED BY id-certUpdateRsp }

The delegator shall use the certUpdateRsp content type to report the outcome of the updates to status information on public-key certificates.

CertUpdateRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertUpdateOK,

failure [1] CertUpdateErr,

... },

... }

CertUpdateOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CertUpdateErr ::= SEQUENCE {

code CASP-error,

... }

The CertUpdateRsp data type specifies the actual content and has the following components:

The CASPcommonComponents data type as specified in clause 21.6.2.

The result component has following two alternatives:

a) The success alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the CertUpdateOK data type.

b) The failure alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the CertUpdateErr data type. The CASP-error data type is specified in clause 21.6.8.

The CertUpdateOK includes an element for each public-key certificate specified in the request in the same order. Each of the element has two alternatives:

a) The ok alternative shall be taken when the update to the public-key certificate information was successfully processed. It has the following components:

– the subject component shall hold the name of the subject to which the public-key certificate had been issued;

– the serialNumber component shall hold the serial number for the public-key certificate.

b) The not-ok alternative shall be taken when a corresponding public-key certificate was not identified.

– the no-reason status code shall be returned when no code is applicable;

– the unknownCert status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

### 21.6.7 Certification authority subscription reject

rejectCAsubscribe CONTENT-TYPE ::= {

RejectCAsubscribe

IDENTIFIED BY id-rejectCAsubscribe }

The rejectCAsubscribe content type is used by receiver of a response content to report problems with the response.

RejectCAsubscribe ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

reason CASP-error,

... }

The RejectCAsubscribe data type specifies the actual content and has the following component:

The sequence component of the CASPcommonComponents data type shall take the same value as in the response on which it is reporting.

The CASP-error is specified in clause 21.5.8.

The delegator shall verify the validity of a received response by checking

a) as specified in clause 21.4.

### 21.6.8 Certification authority subscription error codes

CASP-error ::= ENUMERATED {

noReason (0),

unknownContentType (1),

unsupportedCASPversion (2),

missingContent (3),

missingContentComponent (4),

invalidContentComponent (5),

sequenceError (6),

unknownCertStatus (7),

... }

A value of the CASP-error data type indicates the result of an issued request.

a) the noReason value shall be selected when no other error code is applicable;

b) the unknownContentType value shall be selected if the content type is not known by the receiver;

c) the unsupportedCASPversion value shall be selected if a request or response content specified a CASP version not supported;

d) the unsupportedContentVersion value shall be selected when a request or response includes an unsupported content type;

e) the missingContent value shall be selected when the request or response did not include a content;

f) the missingContentComponent value shall be selected when a request or response does not includes a mandatory component;

g) the invalidContentComponent value shall be selected when an unexpected component was included in a request or response;

h) the sequenceError value shall be selected by when:

– a delegator or a CA receives a request content for the first time that did not have the sequence component set to 1;

– a delegator or a CA receives a request content that did not have the sequence component set to one higher than for a previous request content in the same direction; or

– a delegator or a CA receives a response content with a sequence component value different from the one in the corresponding request content;

*Add new module to the end of Annex A:*

-- A.4 - CMS content specifications module

CmsContentSpecifications {joint-iso-itu-t ds(5) module(1) cmsContentSpecifications(40) 8}

DEFINITIONS ::=

BEGIN

-- EXPORTS All

IMPORTS

-- from Rec. ITU-T X.501 | ISO/IEC 9594-2

authenticationFramework, id-cmsct, informationFramework, algorithmObjectIdentifiers

FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

Name

FROM InformationFramework informationFramework

-- from Rec. ITU-T X.509 | ISO/IEC 9594-8

ALGORITHM, AlgorithmIdentifier{}, Certificate, CertificateSerialNumber, CertWhitelist,

CertStatus, ENCRYPTED-HASH{}, PKCertIdentifier, SIGNATURE{}, TBSCertWhitelist,

Version, WhitelistSerialNumber

FROM AuthenticationFramework authenticationFramework

sha256, sha224, sha256WithRSAEncryptionAlgorithm

FROM AlgorithmObjectIdentifiers algorithmObjectIdentifiers

Attributes{},

CMSVersion, id-signedData, RevocationInfoChoices, SignatureValue,

SignedAttributes, UnsignedAttributes

FROM CMS {itu-t recommendation(0) x(24) cms(894) module(0) version1(1)} ;

-- Signed data adapted

wlSignedData CONTENT-TYPE ::= {

WLSignedData

IDENTIFIED BY id-signedData }

WLSignedData ::= SEQUENCE {

version CMSVersion (v3),

digestAlgorithms SET (SIZE (1)) OF AlgorithmIdentifier {{WL-Hash-Algorithms}},

encapContentInfo EncapsulatedContentInfo,

certificates [0] IMPLICIT SET (SIZE (1..MAX)) OF Certificate OPTIONAL,

--crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,

signerInfos SignerInfos,

... }

EncapsulatedContentInfo ::= SEQUENCE {

eContentType CONTENT-TYPE.&id({WLContentSet}),

eContent [0] EXPLICIT OCTET STRING

(CONTAINING CONTENT-TYPE.&Type({WLContentSet}{@eContentType})) OPTIONAL }

SignerInfos ::= SET (SIZE (1)) OF SignerInfo

SignerInfo ::= SEQUENCE {

version CMSVersion,

sid SignerIdentifier,

digestAlgorithm AlgorithmIdentifier {{WL-Hash-Algorithms}},

signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,

signatureAlgorithm AlgorithmIdentifier {{WL-Signature-Algorithms}},

signature SignatureValue,

unsignedAttrs [1] IMPLICIT Attributes{{UnsignedAttributes}} }

SignerIdentifier ::= CHOICE {

--issuerAndSerialNumber IssuerAndSerialNumber,

subjectKeyIdentifier [0] SubjectKeyIdentifier,

--certHash [1] CertHash,

...}

SubjectKeyIdentifier ::= OCTET STRING

WL-Hash-Algorithms ALGORITHM ::= {sha256 | sha224, ...}

WL-Signature-Algorithms ALGORITHM ::= {sha256WithRSAEncryptionAlgorithm, ...}

-- CMS content types

CONTENT-TYPE ::= TYPE-IDENTIFIER

WLContentSet CONTENT-TYPE ::= {

addWhitelistReq |

addWhitelistRsp |

replaceWhitelistReq |

replaceWhitelistRsp |

updateWhitelistReq |

updateWhitelistRsp |

deleteWhitelistReq |

deleteWhitelistReq |

rejectWhitelist |

certSubscribeReq |

certSubscribeRsp |

certUnsubscribeReq |

certUnsubscribeRsp |

certReplaceReq |

certReplaceRsp |

certUpdateReq |

certUpdateRsp |

rejectCAsubscribe,

... }

-- Whitelist management

WLMPcommonComponents ::= SEQUENCE {

version WLMPversion DEFAULT v1,

sequence WLMPsequence,

... }

WLMPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

WLMPsequence ::= INTEGER (1..MAX)

addWhitelistReq CONTENT-TYPE ::= {

AddWhitelistReq

IDENTIFIED BY id-addWhitelistReq }

AddWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

certlist CertWhitelist,

... }

addWhitelistRsp CONTENT-TYPE ::= {

AddWhitelistRsp

IDENTIFIED BY id-addWhitelistRsp }

AddWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] AddWhitelistOK,

failure [1] AddWhitelistErr,

... },

... }

AddWhitelistOK ::= SEQUENCE {

ok NULL,

... }

AddWhitelistErr ::= SEQUENCE {

notOK WLMP-error,

... }

replaceWhitelistReq CONTENT-TYPE ::= {

ReplaceWhitelistReq

IDENTIFIED BY id-replaceWhitelistReq }

ReplaceWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

old WhitelistSerialNumber OPTIONAL,

new CertWhitelist,

... }

replaceWhitelistRsp CONTENT-TYPE ::= {

ReplaceWhitelistRsp

IDENTIFIED BY id-replaceWhitelistRsp }

ReplaceWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] RepWhitelistOK,

failure [1] RepWhitelistErr,

... },

... }

RepWhitelistOK ::= SEQUENCE {

ok NULL,

... }

RepWhitelistErr ::= SEQUENCE {

notOK WLMP-error,

... }

updateWhitelistReq CONTENT-TYPE ::= {

UpdateWhitelistReq

IDENTIFIED BY id-updateWhitelistReq }

UpdateWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

wl-Id WhitelistSerialNumber OPTIONAL,

status SEQUENCE (SIZE (1..MAX)) OF WhitelistStatus,

signature WLsignature,

... }

WhitelistStatus ::= SEQUENCE {

subjectId PKCertIdentifier,

update CertStatus,

... }

WLsignature ::= ENCRYPTED-HASH {TBSCertWhitelist}

updateWhitelistRsp CONTENT-TYPE ::= {

UpdateWhitelistRsp

IDENTIFIED BY id-updateWhitelistRsp }

UpdateWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] UpdWhitelistOK,

failure [1] UpdWhitelistErr,

... },

... }

UpdWhitelistOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

result NULL,

... },

not-ok [1] SEQUENCE {

status WLMP-CertStatusCode,

... },

... }

WLMP-CertStatusCode ::= ENUMERATED {

noReason (1),

unknownCert (2),

unknownCertStatus (3),

... }

UpdWhitelistErr ::= SEQUENCE {

code WLMP-error,

... }

deleteWhitelistReq CONTENT-TYPE ::= {

DeleteWhitelistReq

IDENTIFIED BY id-deleteWhitelistReq }

DeleteWhitelistReq ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

wl-Id WhitelistSerialNumber OPTIONAL,

... }

deleteWhitelistRsp CONTENT-TYPE ::= {

DeleteWhitelistRsp

IDENTIFIED BY id-deleteWhitelistRsp }

DeleteWhitelistRsp ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

result CHOICE {

success [0] DelWhitelistOK,

failure [1] DelWhitelistErr,

... },

... }

DelWhitelistOK ::= SEQUENCE {

ok NULL,

... }

DelWhitelistErr ::= SEQUENCE {

notOK WLMP-error,

... }

rejectWhitelist CONTENT-TYPE ::= {

RejectWhitelist

IDENTIFIED BY id-rejectWhitelist }

RejectWhitelist ::= SEQUENCE {

COMPONENTS OF WLMPcommonComponents,

reason WLMP-error,

... }

WLMP-error ::= ENUMERATED {

noReason (0),

unknownContentType (1),

unsupportedWLMPversion (2),

missingContent (3),

missingContentComponent (4),

invalidContentComponent (5),

sequenceError (6),

invalidSignature (7),

duplicateWL (8),

missingWLComponent (9),

invalidWLversion (10),

constraintError (11),

unknownCertStatus (12),

unsupportedCriticalExtenssion (13),

maxWLsExceeded (14),

unknownCert (15),

unknownWL (16),

... }

-- CA subscription

CASPcommonComponents ::= SEQUENCE {

version CASPversion DEFAULT v1,

sequence CASPsequence,

... }

CASPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

CASPsequence ::= INTEGER (1..MAX)

certSubscribeReq CONTENT-TYPE ::= {

CertSubscribeReq

IDENTIFIED BY id-certSubscribeReq }

CertSubscribeReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

... }

certSubscribeRsp CONTENT-TYPE ::= {

CertSubscribeRsp

IDENTIFIED BY id-certSubscribeRsp }

CertSubscribeRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertSubscribeOK,

failure [1] CertSubscribeErr,

... },

... }

CertSubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

cert Certificate,

status CertStatus,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CASP-CertStatusCode ::= ENUMERATED {

noReason (1),

unknownCert (2),

... }

CertSubscribeErr ::= SEQUENCE {

code CASP-error,

... }

certUnsubscribeReq CONTENT-TYPE ::= {

CertUnsubscribeReq

IDENTIFIED BY id-certUnsubscribeReq }

CertUnsubscribeReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

... }

certUnsubscribeRsp CONTENT-TYPE ::= {

CertUnsubscribeRsp

IDENTIFIED BY id-certUnsubscribeRsp }

CertUnsubscribeRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertUnsubscribeOK,

failure [1] CertUnsubscribeErr,

... },

... }

CertUnsubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CertUnsubscribeErr ::= SEQUENCE {

code CASP-error,

... }

certReplaceReq CONTENT-TYPE ::= {

CertReplaceReq

IDENTIFIED BY id-certReplaceReq }

CertReplaceReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {

old CertificateSerialNumber,

new Certificate,

... },

... }

certReplaceRsp CONTENT-TYPE ::= {

CertReplaceRsp

IDENTIFIED BY id-certReplaceRsp }

CertReplaceRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertReplaceOK,

failure [1] CertReplaceErr,

... },

... }

CertReplaceOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

issuer Name,

serialNumber CertificateSerialNumber,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CertReplaceErr ::= SEQUENCE {

code CASP-error,

... }

certUpdateReq CONTENT-TYPE ::= {

CertUpdateReq

IDENTIFIED BY id-certUpdateReq }

CertUpdateReq ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

certs SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

certStatus CertStatus,

... },

... }

certUpdateRsp CONTENT-TYPE ::= {

CertUpdateRsp

IDENTIFIED BY id-certUpdateRsp }

CertUpdateRsp ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

result CHOICE {

success [0] CertUpdateOK,

failure [1] CertUpdateErr,

... },

... }

CertUpdateOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {

ok [0] SEQUENCE {

subject Name,

serialNumber CertificateSerialNumber,

... },

not-ok [1] SEQUENCE {

status CASP-CertStatusCode,

... },

... }

CertUpdateErr ::= SEQUENCE {

code CASP-error,

... }

rejectCAsubscribe CONTENT-TYPE ::= {

RejectCAsubscribe

IDENTIFIED BY id-rejectCAsubscribe }

RejectCAsubscribe ::= SEQUENCE {

COMPONENTS OF CASPcommonComponents,

reason CASP-error,

... }

CASP-error ::= ENUMERATED {

noReason (0),

unknownContentType (1),

unsupportedWLMPversion (2),

missingContent (3),

missingContentComponent (4),

invalidContentComponent (5),

sequenceError (6),

unknownSubject (7),

unknownCert (8),

... }

id-addWhitelistReq OBJECT IDENTIFIER ::= {id-cmsct 0}

id-addWhitelistRsp OBJECT IDENTIFIER ::= {id-cmsct 1}

id-replaceWhitelistReq OBJECT IDENTIFIER ::= {id-cmsct 2}

id-replaceWhitelistRsp OBJECT IDENTIFIER ::= {id-cmsct 3}

id-updateWhitelistReq OBJECT IDENTIFIER ::= {id-cmsct 4}

id-updateWhitelistRsp OBJECT IDENTIFIER ::= {id-cmsct 5}

id-deleteWhitelistReq OBJECT IDENTIFIER ::= {id-cmsct 6}

id-deleteWhitelistRsp OBJECT IDENTIFIER ::= {id-cmsct 7}

id-rejectWhitelist OBJECT IDENTIFIER ::= {id-cmsct 8}

id-certSubscribeReq OBJECT IDENTIFIER ::= {id-cmsct 9}

id-certSubscribeRsp OBJECT IDENTIFIER ::= {id-cmsct 10}

id-certUnsubscribeReq OBJECT IDENTIFIER ::= {id-cmsct 11}

id-certUnsubscribeRsp OBJECT IDENTIFIER ::= {id-cmsct 12}

id-certReplaceReq OBJECT IDENTIFIER ::= {id-cmsct 13}

id-certReplaceRsp OBJECT IDENTIFIER ::= {id-cmsct 14}

id-certUpdateReq OBJECT IDENTIFIER ::= {id-cmsct 15}

id-certUpdateRsp OBJECT IDENTIFIER ::= {id-cmsct 16}

id-rejectCAsubscribe OBJECT IDENTIFIER ::= {id-cmsct 17}

END -- CmsContentSpecifications