MISPC
Minimum Interoperability Specification for PKI Components, Version 1

September 3, 1997

William Burr, **Donna Dodson**, **Noel Nazario**, **W. Timothy Polk**

Output of NIST's Cooperative Research and Development Agreements for Public Key Infrastructure development with AT&T, BBN, Certicom, Cylink, DynCorp, IRE, Motorola, Northern Telecom, Spyrus, and VeriSign.

Table of Contents

Acknowledgements

This document was produced by NIST in cooperation with ten industry partners through Cooperative Research and Development Agreements (CRADAs.) The participating companies were:

- AT&T;
- BBN;
- Certicom;
- Cylink;
- DynCorp;
- Northern Telecom;
- IRE;
- Motorola;
- Spyrus, Inc.; and
- VeriSign, Inc.

The authors wish to acknowledge the important contributions of our CRADA partners. Their insight in both technical and business issues was invaluable to this project. Without their assistance, this document would not exist.

The authors also wish to acknowledge the contribution of Nelson Hastings, who performed a thorough and detailed review of this specification.

Minimum Interoperability Specification for PKI Components

1. Introduction

1.1 Purpose

The Minimum Interoperability Specification for PKI Components (MISPC) provides a basis for interoperation between public key infrastructure (PKI) components from different vendors. This specification will be available to companies interested in offering interoperable PKI components, to Federal agencies developing procurement specifications, and to other interested parties. It will be the basis for a NIST reference implementation and an initial root CA for the Federal PKI. A test suite for conformance to this specification is also planned.

1.2 Scope

This specification supports interoperability for a large scale PKI that issues, revokes and manages digital signature public key certificates, to allow the use of those signatures to replace handwritten signatures in government services, commerce, and legal proceedings, and to allow distant parties, who have no previous relationship, to reliably authenticate each other and conduct business. Such a PKI, and the certificates it requires, may be excessive for some applications, and other more streamlined certificates and protocols may be more appropriate for more specialized and restricted applications.

It is recognized that the PKI will simultaneously support certificates for confidentiality key management, however that is outside the scope of this specification. A sound digital signature PKI should provide the basic foundation needed for issuing any kind of public key certificate, including key management certificates, and it is anticipated that confidentiality key management will be addressed in a future revision.

The MISPC addresses:

- public key certificate generation, renewal, and revocation;
- signature generation and verification; and,
- certificate and certification path validation.

The specification consists primarily of a profile of certificate and CRL extensions and a set of transactions. The transactions include: certification requests, certificate renewal, certificate revocation, and retrieval of certificates and CRLs from repositories.

The MISPC focuses primarily on the aspects of PKI interoperation most apparent to end users, that is how to request and be issued a certificate, how to sign documents, how to retrieve the certificates of others, and how to validate signatures. Some aspects of the "internal" operation of a PKI, as outlined below, have not reached sufficent stability at this point, and are therefore not specified.

In this specification a PKI is broken into five components:

- *Certification Authorities (CAs)* that issue and revoke certificates;

- *Organizational Registration Authorities (ORAs)* that vouch for the binding between public keys and certificate holder identities and other attributes;
- *Certificate holders* that are issued certificates and can sign digital documents;
- *Clients* that validate digital signatures and their certification paths from a known public key of a trusted CA;
- *Repositories* that store and make available certificates and Certificate Revocation Lists (CRLs).

Many entities will include certificate holder and client functionality. CAs and ORAs will include both certificate holder and client functionality. End-entity certificate holders will generally also have client functionality. There may be some clients, however, that are not also certificate holders.

Repositories are not necessarily certificate holders and may not include client functionality. This interoperability specification addresses only one aspect of repositories, the protocol used by clients to request certificates and CRLs from the repository. This is because the precise concept, role and business model of repositories is unsettled. The X.509 certificate standard [ISO94-8] itself assumes the existence of an X.500 directory, to satisfy repository requirements, however X.500 directories, while available for some time, have not been, and do not appear to be going to be widely used.

The MISPC specifies the Lightweight Directory Access Protocol (LDAP) version 2 as the vehicle for client access of repositories, primarily because it appears to be the most generally accepted and broadly implemented alternative. This choice does not address, for example, standardized protocols for CAs to use to update repositories, nor does it address protocols for repositories to automatically shadow one another, both of which may be desirable. The former can be addressed on a case by case base between CAs and their repositories, and the latter may not be necessary.

In the conventional approach to certificate status confirmation (which the MISPC follows), repositories are not trusted entities, rather it is the CA's signature on a CRL that validates the revocation status of certificates. On-line mechanisms for real-time certificate status confirmation would require that repositories themselves be trusted entities and that they authenticate themselves to clients. Standardized protocols for such certificate status confirmation are not yet available. Therefore such protocols are outside the scope of this specification, but, since real-time certificate status confirmation may be needed for some applications, this subject may be addressed in a later revision.

The MISPC does not include a protocol for repositories to authenticate users, which would be needed to implement access by access billing for repository use. Although that may become an important business model for repositories, there does not currently appear to be enough agreement on such a business model and the supporting protocol to make this subject ripe for inclusion in a minimum interoperability specification. This subject may also be addressed in a later revision.

In some cases, out-of-band exchanges must be performed as part of the transactions defined by this specification. The format and contents of such out-of-band transactions are generally outside of the scope of this specification.[1]

1.3 Approach

The MISPC is based on X.509 version 3 certificates and version 2 CRLs. To the extent possible, this document adopts data formats and transaction sets defined in existing and evolving standards such as ITU-T X.509 [ISO94-8], ANSI [X9.55], [X9.57], and [X9.62] and the IETF's PKIX working documents [PKIX1], [PKIX3]. In drafting this document, whenever the stability of an evolving standard used in this document has come to question, NIST has made an educated guess regarding the direction to be followed. These issues were reviewed by industry collaborators prior to the release of this specification and represented vigorously within the appropriate standards groups to minimize departure from the stable version of the standards.

1.4 Assumptions

The MISPC assumes that CAs, ORAs, and certificate holders are physically separated. Where these entities are physically collocated, support for specified interfaces is not required. In particular, a PKI component that includes both ORA and CA functionality is not required to support the MISPC message formats for transactions between these components. However, if that system includes a CA that supports remote ORAs in addition to the local ORA function, it must support the MISPC transactions for the remote ORAs.

The MISPC considers CAs and ORAs as functional entities in a PKI. The internal design of these entities is outside the scope of this specification.

The MISPC identifies three important digital signature algorithms for which suitable approved or mature draft standards exist. New algorithms could easily be incorporated as they are introduced in standards.

The MISPC supports both hierarchical and networked trust models [CONOPS]. In hierarchical models, trust is delegated by a CA when it certifies a subordinate CA. Trust delegation starts at a root CA that is trusted by every node in the infrastructure. In network models, trust is established between any two CAs in peer relationships (cross-certification), thus allowing the possibility of multiple trust paths between any two CAs. The MISPC assumes that X.509 v3 extensions, such as **basicConstraints**, **nameConstraints, keyUsage,** and **certificatePolicy**, will be included in certificates to explicitly manage trust relationships.

The MISPC assumes that certificates and certificate revocation lists (CRLs) will be available in a repository for retrieval without authentication. MISPC clients will perform path validation by obtaining the necessary certificates and CRLs from the appropriate repositories. The repository may be an X.500 directory or some other type accessible by using Universal Resource Identifier (URI) notation. Repositories are expected to support the Lightweight Directory Access Protocol (LDAP) [RFC 1777], therefore compliant products are required to support this protocol.

---

[1] The format and content of the electronic data provided to an ORA when requesting a certificate "in-person" is the exception to this rule. See section 3.5.1, *ORA-Generated Registration Requests*

These repositories need not be linked together and other protocols may be used to retrieve certificates and CRLs. The specification requires explicit identification of the certificate repositories used and retrieval mechanisms for the issuer's certificate(s) and CRLs within the certificate.[2]

Certificate Revocation Lists (CRLs) are expected to be a widely implemented mechanism for revoking and validating the status of unexpired certificates. While the use of CRLs for this purpose may not be universal, and some CAs may choose to provide an on-line mechanism for validating certificate status in real time, CRL generation will be necessary for interoperability with users of other CAs. In addition to current checks of certificate validity, CRLs provide an important mechanism for documenting the historical revocation status of certificates. That is, a dated signature may be presumed to be valid if the signature date were within the validity period of the certificate, and the current CRL of the issuing CA at that date did not show the certificate to be revoked.[3]

Therefore, the MISPC assumes that CA products will be able to generate CRLs, and that clients will be able to use CRLs when validating certificates.

1.5 Definitions, Terms, and Acronyms

Abstract Syntax Notation 1 (ASN.1): an abstract notation for structuring complex data objects.

accredit: recognize an entity or person to perform a specific action; CAs accredit ORAs to act as their intermediary (see organizational registration authority below).

certificate (or public key certificate): A digitally signed data structure defined in the X.509 standard [ISO94-8] that binds the identity of a certificate holder (or subject) to a public key.

certificate holder: An entity that is named as the subject of a valid certificate.

certificate policy: A named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements. For example, a particular certificate policy might indicate applicability of a type of certificate to the authentication of electronic data interchange transactions for the trading of goods within a given price range.

certificate user: An entity that uses certificates to know, with certainty, the public key of another entity.

certificate-using system: An implementation of those functions defined in the X.509 standard [ISO94-8] that are used by a certificate user. This term is defined in the Draft Amendments to X.509 [DAM] and equivalent to the term "client" used in this interoperability specification.

Certification Authority (CA): A trusted entity that issues certificates to end entities and other CAs. CAs issue CRLs periodically, and post certificates and CRLs to a repository.

---

[2] As a consequence of this assumption, the distinguished name of the subject is not sufficient to retrieve a certificate. MISPC clients must obtain the signer's certificate, or distinguished name of the subject and the identity of the repository, from the signer.

[3] This assumes you can accept the date attached to the signature on the basis of a trusted archive or notarization, which are outside the scope of this specification.

certification path: An ordered sequence of certificates, leading from a certificate whose public key is known by a client, to a certificate whose public key is to be validated by the client.

Certification Practice Statement: A statement of the practices which a Certification Authority employs in issuing certificates.

CRL distribution point: A directory entry or other distribution source for CRLs; a CRL distributed through a CRL distribution point may contain revocation entries for only a subset of the full set of certificates issued by one CA or may contain revocation entries for multiple CAs.

certificate revocation list (CRL): a list of revoked but unexpired certificates issued by a CA.

certify: the act of issuing a certificate.

client (or PKI client): A function that uses the PKI to obtain certificates and validate certificates and signatures. Client functions are present in CAs and end entities. Client functions may also be present in entities that are not certificate holders. That is, a system or user that verifies signatures and validation paths is a client, even if it does not hold a certificate itself. See section 2.4.

delta-CRL: A partial CRL indicating only changes since a prior CRL issue.

DES: The symmetric encryption algorithm defined by the Data Encryption Standard (FIPS 46-2).

DES MAC: An algorithm for generating a message authentication code (MAC) using the symmetric encryption algorithm DES.

Distinguished Encoding Rules (DER): rules for encoding ASN.1 objects which give a consistent encoding for each ASN.1 value. Implementations conforming to this specification shall encode ASN.1 objects using the DER.

digital signature: a data unit that allows a recipient of a message to verify the identity of the signatory and integrity of the message.

Digital Signature Algorithm (DSA): the digital signature algorithm specified in FIPS PUB 186.

directory service (DS): a distributed database service capable of storing information, such as certificates and CRLs, in various nodes or servers distributed across a network.

end entity: A certificate subject which uses its private key for purposes other than signing certificates.

Elliptic Curve Digital Signature Algorithm (ECDSA): a digital signature algorithm that is an analog of DSA using elliptic curve mathematics and specified in ANSI draft standard X9.62 [X9.62].

hash: a function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties: it is computationally infeasible to find for a given output an input which maps to this output; and it is computationally infeasible to find for a given input a second input which maps to the same output.

hash code: The string of bits which is the output of a hash function

LDAP: The Lightweight Directory Access Protocol, or LDAP, is a directory access protocol. In this document, LDAP refers to the protocol defined by RFC 1777, which is also known as LDAP V2. LDAP V2 describes unauthenticated retrieval mechanisms.

message authentication code: a data authenticator generated from the message, usually through cryptographic techniques. In general, a cryptographic key is also required as an input.

message digest: the fixed size result of hashing a message.

Organizational Registration Authority (ORA): an entity that acts an intermediary between the CA and a prospective certificate subject; the CA trusts the ORA to verify the subject's identity and that the subject possesses the private key corresponding to the public key to be bound to that identity in a certificate. Note that equivalent functions are referred to as Local Registration Authority (LRAs) or Registration Authorities (RAs) in some documents.

out of band: Some transactions between PKI components will be performed through physical procedures rather than implemented electronically. Such transactions are described as out-of-band transactions.

policy mapping: Recognizing that, when a CA in one domain certifies a CA in another domain, a particular certificate policy in the second domain may be considered by the authority of the first domain to be equivalent (but not necessarily identical in all respects) to a particular certificate policy in the first domain.

repository: a database service capable of storing information, such as certificates and CRLs, allowing unauthenticated information retrieval. Repositories include, but are not limited to, directory services.

RSA: For the purposes of this specification, RSA is a public-key signature algorithm specified by PKCS #1 [PKCS#1]. As a reversible public-key algorithm, it may also be used for encryption.

URI: A uniform resource identifier, or URI, is a short string containing a name or address which refers to an object in the "web."

URL: A uniform resource locator, or URL, is a short string containing an address which refers to an object in the "web." URLs are a subset of URIs.

Well Known X.500 Directory: In some environments, an X.500 service may be widely available and used throughout an organization. If such a directory service is used to distribute certificates and CRLs issued by that organization, such information need not be included in the certificate.

2. Infrastructure Component Specifications

This section specifies a minimal set of functions and transactions required for the interoperation of PKI components. It includes specifications for CAs, ORAs, and PKI Clients.

2.1 Certification Authority (CA)

CAs generate, revoke, publish, and archive certificates. They rely upon a repository to make certificates and CRLs available to all certificate users.

To enable CAs to join existing hierarchically managed infrastructures, they shall be able to request certificates from a parent CA. CAs shall also be able to generate cross certificates, to support cross-certification with other CAs as allowed by their policies.

CAs accredit ORAs, which vouch for the identity and other attributes of users requesting certificates. This accreditation is an off-line decision to accept ORA-generated certification requests from that ORA. CAs identify certificate holders using X.500 distinguished names. Distinguished names uniquely identify certificate holders.

CAs themselves include both a certificate holder function to request, revoke and renew certificates issued by other CAs (see sec. 2.3) and a client function to retrieve certificates and CRLs, and validate certification paths (see sec. 2.4).

2.1.1 Interoperability-Relevant CA Functional Specifications

CAs perform the following functions:

- Issue and deliver subordinate and cross certificates;
- Accept revocation requests from certificate holders and ORAs for certificates it issued;
- Post certificates and CRLs to the repository; and
- Request CA certificates.

Issuing Certificates

CAs support three types of certification requests: *self-registration*, *ORA-generated registration*, and *renewal*.[4] CAs authenticate the identity of the certificate's subject differently for each type of request. The prospective certificate holder supplies an authenticator in a *self-registration request*; the authenticator is derived from a secret obtained from an ORA. ORAs generate and sign *ORA-generated registration requests*, vouching for the identity of the subject, when the subject physically attends the ORA. The subjects of currently valid certificates can vouch for their own identity in a *renewal request* by signing with their current private key.

In an ORA-generated registration request, the ORA vouches for the prospective certificate holder's identity and the binding to the public key. When CAs receive certification requests from accredited ORAs, they shall process the requests and, if accepted, generate new certificates, post the certificates to a repository[5], and send them to the requesting ORAs. CAs may also send

[4] CAs may be configurable to reject one or more classes of certification requests if the certificate policy prohibits such transactions.

[5] Conforming CAs shall be able to post the certificates they issue to a repository. However, it is not necessary to post end-entity certificates, since the certificate holder may provide the certificate with the signed document.

the new certificate to the certificate holders. CAs shall reject ORA-generated certification requests that do not come from a recognized ORA, that have invalid signatures, or that contain unmatched information. If a CA rejects an ORA-generated certification request, it shall report the failure to the ORA stating the reason.

In a self-registration request, the ORA provides a secret message to the prospective certificate holder. The entity generates its own key pair, forms a certification request, signs it with the corresponding private key material, and includes authentication information based on the secret provided by the ORA.[6] The CA receives the request, verifies the requester's identity through the authentication information and verifies the entity holds the corresponding private key material. If accepted, the CA will generate a new certificate, post the certificate to the repository, and send it to the certificate holder. The CA may reject self-registration requests if the authentication information does not verify, the signature is invalid, or fields contain unmatched information. If a CA rejects a self-registration request, it shall report the failure to the requester stating the reason.

In a renewal request, the established identity of the requester is perpetuated with the request. Certificate renewals are initiated by the certificate holder and sent directly to the CA. CAs process the renewals and, if correct, send the new certificates to the certificate holders and post the new certificates to the repository. CAs may reject certificate renewal requests with invalid signatures, requests from entities not currently certified, and renewal requests that are not allowed by the CA's certification practice statement or the certificate policy. If a CA rejects a certificate renewal request, it shall report the failure to the requesting entity stating the reason.

<u>Cross Certification</u>

CAs may issue certificates to other CAs with appropriate constraints. The decision to cross-certify is made out-of-band and involves examination of Certification Practice Statements and certificate policies. Each CA determines the appropriate constraints for path validation by their users. After obtaining the other CA's public key, the CA generates the certificate and posts it to the repository.

Optionally, cross-certifying CAs may exchange certificates, construct certificate-pairs, and post them to the repository.[7]

<u>Revoking Certificates</u>

CAs shall be capable of generating and issuing certificate revocation lists (CRLs). CAs shall be able to issue CRLs that contain all revoked certificates that they issued and have not expired. Optionally, CAs may also issue indirect and delta CRLs. The types of CRLs issued will be determined by the CA's certification practice statement.

In those cases where a CA issues a single CRL for all revoked certificates it has issued:

---

[6] Where the CA and ORA are not co-located, this also requires an exchange of secrets between the CA and ORA. Details of this exchange are outside the scope of this specification.

[7] A CA may issue a certificate to another CA even if that latter refuses to issue a certificate to the former. In this case, the CA could (optionally) construct a cross certificate pair containing only the **reverseCertificate**.

- When a new CRL is generated, all revoked unexpired certificates from the previous CRL shall be carried over to the new CRL, and any certificates with approved pending certificate revocation requests shall be added to the new CRL. Certificates on the previous CRL with a reason code of **certificateHold** may be carried over to the new CRL, revoked on the new CRL, or omitted from the new CRL. Omission from the new CRL indicates the CA will vouch for the binding between the subject and public key. A certificate with an approved pending certificate revocation request shall be included in the next CRL even if it expires before the CRL is issued.
- In this case, CAs shall only revoke certificates they issued.[8] The signer of the revocation request must either be the certificate holder or an authorized entity (such as an accredited ORA) acting on behalf of the certificate holder or the certificate holder's organization. CAs shall validate revocation requests prior to including a certificate in a CRL. Validation of a revocation request shall include validation of the signature on the request. Out-of-band validation of revocation requests signed by ORAs may optionally be required by the certificate policy.

CAs shall issue X.509 version 2 CRLs.[9] The fields and extensions utilized, and the values assigned to them, shall be in accordance with section 3.2.1. After generating and signing a CRL, CAs shall send it to the repository.

Post Certificates, Cross Certificates, and CRLs

CAs shall be capable of posting certificates, cross certificate pairs, and CRLs for retrieval by PKI clients. CAs shall always post CA certificates, cross certificate pairs, and CRLs. Posting of end-entity certificates is optional. The mechanisms used to update directories is beyond the scope of this specification.

Request CA Certificates

CAs shall be capable of requesting certificates from hierarchically superior CAs to support PKIs based on the hierarchical trust model. This request is supported as described in section 3.5.1. The certificate request shall identify the entity as a CA through the **basicConstraints** extension as described in section 3.1.3.3.

2.1.2 Electronic Transaction Set.

Table 2-1 summarizes electronic transactions used in providing certificate management services. These transactions enable:

- processing of certification requests and certificate revocation requests for end entity certificates;
- posting of certificates and CRLs on the repository;
- the retrieval of certificates and CRLs from the repository for signature validation.

[8] Revocation may be initiated by receipt of a signed request, or by the CA's own procedures. This specification does not address revocations initiated by the CA.

[9] Version 2 CRLs correspond to the Version 3 certificate; the Version 2 certificate definition did not result in creation of a new CRL format.

CAs shall process ORA-generated certification requests in the form of **CertReq** messages.[10] **CertReq** messages are signed by the ORA in the **PKIProtection** structure. By signing requests, ORAs vouch for the identity of the certificate holder and confirm that requesting certificate holders are in possession of the corresponding private keys. CAs respond to the ORAs or certificate holders with **CertRep** messages. If a request was accepted, the **CertRep** message contains the new certificate. If the request was rejected, the message contains the error code (see sec. 3.5.1).

CAs shall also support the self-registration request, where users who are not current certificate holders sign their own certificate request. The CA shall require the entity to generate authentication information based on out-of-band interaction with an ORA. This information substitutes for ORA signature to vouch for the requester's identity. To request a certificate without appearing before an ORA, the entity obtains some information out-of-band from the ORA. This information might be a symmetric key for use in generation of a MAC or keyed hash. The entity generates a **CertReq** message and signs it with the entity's new private key. This message is then protected with the information obtained out-of-band as directed by the ORA. The CA generates a **CertRep** message; if the request was fulfilled the message contains the new certificate. If the request was rejected, the message contains error codes. This transaction is described in detail in section 3.5.3.[11]

CAs shall process certificate renewal requests in the form of **CertReq** messages. These messages are sent to a CA by the entity requesting the certificate. The message shall include the certificate holder's distinguished name, the serial number of their current certificate, and the new public key. The message may optionally include a proposed validity period and a proposed key id. The message shall be signed with the private key corresponding to the certificate holder's unexpired, unrevoked certificate and the new private key, as described in section 3.5.2. CAs shall respond to the requester in the form of an **CertRep** message. This message shall contain either a new certificate or a failure code. If issued, the certificate shall include the certificate holder's distinguished name and the new public key. CAs are free to modify the validity period proposed in the request. CAs shall generate a key identifier if the message did not include one.

CAs shall receive **RevReq** messages from ORAs or certificate holders. The **RevReq** message shall include the certificate serial number or the certificate holder's distinguished name and the key identifier. CAs shall respond with a **RevRep** message. This message shall include status and failure information, and may include additional details about the revoked certificate.

CAs shall post CA certificates, cross certificate pairs, and CRLs that it issues to a repository. CAs may optionally be capable of posting end entity certificates to a repository.[12]

---

[10] This section refers to **CertReq**, **CertRep**, **RevReq** and **RevRep** messages. The precise structure and content of these messages is defined in section 3.4.

[11] An alternative syntax for this transaction is specified in section 3.5.4.

[12] Posting of end entity certificates is not strictly required, since the originator of a signature can supply their own certificate.

Table 2-1 CA Electronic Transaction Set

Transaction	Description	From	To
ORA-Generated Registration Request (see sec. 3.5.1)	ORA submits a certificate request on behalf of an authenticated entity	ORA	CA
	CA returns signed certificate or error message	CA	ORA and optionally, certificate holder
Certificate Revocation (sec. 3.5.5)	ORA or certificate holder requests revocation of a certificate	ORA or certificate holder	Issuer CA
	CA responds with acceptance or rejection of the revocation request	Issuer CA	ORA or certificate holder
Self-Registration Request (secs. 3.5.3 and 3.5.4)	message signed with new public key encapsulates certificate request with ORA-directed protection value	client	Issuer CA
	CA returns signed certificate and CA's certificate or an error message	Issuer CA	client
Certificate Renewal Request (sec. 3.5.2)	certificate request containing new public key with proof of possession and current certificate serial number; signed with current private key	certificate holder	CA
	CA returns signed certificate or error message	CA	certificate holder

2.2 Organizational Registration Authority (ORA)

ORAs vouch for the identity of entities requesting certification. ORAs may verify that identity by requiring the requesting entity to attend the ORA physically with a physical token, or through out-of-band mechanisms. Where the entity physically attends the ORA, the ORA also verifies their possession of private key material corresponding to the public key by verifying a signed message (as described in sec. 3.5.1).

The format for a certificate request on behalf of an entity in physical attendance appears in section 3.5.1. ORAs shall verify the entity possesses a complete key pair. After the key pair and the entity's identity are verified, an ORA signs and sends an electronic certificate request to the appropriate CA.

Certificate requests on behalf of a user who does not physically attend the ORA require that the ORA provide authentication information to the entity. This information is used by the entity to authenticate itself to the CA in a self-registration request as defined in section 3.5.3. This specification does not define the content or format of the out-of-band exchange(s) required to implement self-registration requests.

ORAs may request certificate revocation for end-entity certificates issued by CAs that have accredited them. The format of the RevReq is presented in section 3.5.5. The ORA function may be collocated with the CA or performed at a separate facility.

ORAs themselves include both a certificate holder function to request, revoke and renew certificates (where it is the subject) issued by CAs (see sec. 2.3) and a client function to retrieve certificates and CRLs and validate certification paths (see sec. 2.4).

2.2.1 Interoperability-Relevant ORA Functional Specifications

ORAs shall perform the following functions:

- Accept and validate certification requests;
- Send certification requests to the CA;
- Retrieve certificates and CRLs from the repository; and
- Generate certificate revocation requests.

The ORA shall be able to pass the newly signed certificate on to the certificate holder, along with the CA's certificate.

ORAs shall generate and sign certificate revocation requests on behalf of certificate holders who no longer possess their private key and suspect compromise.[13] If permitted by the CA's certification practice statement, ORAs shall also generate and sign certificate revocation requests on behalf of the certificate holder's organization. Revocation requests are signed by the ORA which then sends them to the issuing CA.

2.2.2 Transaction Set

Table 2-2 gives the subset of electronic transactions used by ORAs. These transactions enable request, delivery, and revocation of end entity certificates, and the retrieval of certificates and CRLs from the repository for signature validation. The following text provides an overview of these transactions; they are described more fully in section 3.5.

ORAs receive certification requests from prospective certificate holders in the form of **CertReq** messages. The **CertReq** message is signed by the prospective certificate holder in the **PKIProtection** structure. After reviewing the requester's credentials and confirming that the prospective certificate holder is in possession of the corresponding private key, ORAs extract the public key information, and create a new **CertReq** message with the ORA's name and signature. ORAs send this message to a CA. ORAs shall provide certificate holders with the CA's certificate.

---

[13] Signature keys lost but not believed compromised are not necessarily revoked; this is determined by policy. Note that confidentiality keys which are lost must be revoked regardless, or a sending party may encrypt and transmit messages the receiver could never decrypt.

Table 2-2 ORA Electronic Transaction Set

Transaction	Description	From	To
ORA-Generated Registration Request (sec. 3.5.1)	User (or system administrator) submits digitally signed certificate request to ORA with proof of identity	client	ORA
	ORA submits a certificate request on behalf of an authenticated prospective certificate holder	ORA	CA
	CA returns signed certificate or error message	CA	ORA
Certificate Revocation (sec. 3.5.5)	ORA requests revocation of a certificate	ORA	Issuer CA
	CA responds with acceptance or rejection of revocation request	Issuer CA	ORA

ORAs may receive **CertRep** messages from the CA. If a certification request is rejected, the ORA will review the error code from the CA and may submit a new request. If a certification request is accepted, the ORA may provide the new certificate to the certificate holder.

ORAs shall generate revocation requests upon request of certificate holders who no longer possess their private key or the certificate holder's organization. By signing the request, the ORA is vouching for the identity of the requester. ORAs shall generate **RevReq** messages, including the certificate serial number or the certificate holder's distinguished name and the key identifier. The **RevReq** message shall be signed by an ORA. The CA shall respond to the ORA with a **RevReq** message.

This message shall include status and failure information, and may include additional details about the revoked certificate. If the certificate is revoked, the ORA shall provide this information to the requester. If the request is rejected, the ORA will review the error code and may re-formulate the request.

2.3 Certificate Holder Specifications

The PKI provides certificate management functions for certificate holders. Certificate holders include CAs, ORAs and other end entities. End entities may include persons and computing systems (e.g., routers and firewalls) or applications (in addition to CAs and ORAs).

PKI certificate holders generate signatures and support PKI transactions to obtain, revoke and renew their certificates.

2.3.1 Interoperability-Relevant PKI Certificate Holders Functional Specifications

Certificate holders shall be able to:

- generate signatures;
- generate certificate requests;

- request certificate revocation;
- request certificate renewal (optional).

Certificate holders are also PKI clients, and must also meet the specifications defined in section 2.4.

2.3.2 Certificate Holders Transaction Set

Table 2-3 gives the summary of transactions used by certificate holders. These transactions enable certificate holders to obtain certificates and CRLs from the directory service, request revocation of certificates held by the certificate holder (if any) for whom the client acts, and request new certificates. All client transactions are performed with the CA that issued the certificate the client uses, an ORA accredited by that CA, and repositories.

Certificate holders shall be able to request revocation of their own certificates. This transaction is performed with the CA and permits certificate holders to sign their own certificate revocation requests. Certificate holders generate a **RevReq** message for each certificate they wish to revoke and transmit to the issuing CA. The **RevReq** message shall include the reason for revocation. The CA generates a **RevRep** message for each request and transmits it to the certificate holder. This transaction is described in detail in section 3.5.5.

Certificate holders shall be able to generate a **CertReq** message to present to an ORA for in-person authenticated certificate requests. The certificate holder constructs and signs the **CertReq** message, so the ORA can verify the requester holds corresponding private key material.

Certificate holders may also implement the Certificate Renewal Request. This transaction is performed with the CA and permits a certificate holder to sign their own certificate request (i.e., without an ORA verification of identity). CAs shall support this transaction, but its use is determined by the certificate policy. To request a new certificate without appearing before an ORA, the certificate holder generates a **CertReq** message and signs it with both the new and current private keys. The CA generates a **CertRep** message; if the request was fulfilled the message contains the new certificate. If the request was rejected, the message contains error codes. This transaction is described in detail in section 3.5.2.

Certificate holders may also implement the self-registration request to request a certificate when they are not current certificate holders. This transaction is performed with the CA and permits a certificate holder to sign their own certificate request. The CA shall require the entity to generate or include information based on out-of-band interaction with an ORA. This information substitutes for ORA verification of identity. CAs shall support this transaction, but its use is determined by the certificate policy. To request a certificate without appearing before an ORA, the entity obtains some information out-of-band from the ORA. This information might be a secret key for use in MAC generation or a signed message that will simply be included in the request. The entity generates a **CertReq** message and signs it with the entity’s new private key. The entity attaches appropriate protection information to the signed message as directed by the ORA. The CA generates a **CertRep** message; if the request was fulfilled the message contains the

Table 2-3 Certificate Holders Electronic Transaction Set

Transaction	Description	From	To
ORA-Generated Registration (see sec. 3.5.1)	User (or system administrator) submits digitally signed certificate request to ORA with proof of identity	client	ORA
Certificate Revocation (sec. 3.5.5)	certificate holder requests revocation of a certificate	certificate holder	Issuer CA
	CA responds with acceptance or rejection of revocation request	Issuer CA	certificate holder
Self-Registration Request (secs. 3.5.3 and 3.5.4)	message signed with new public key encapsulates certificate request with ORA-directed protection value	client	Issuer CA
	CA returns signed certificate and CA's certificate or an error message	Issuer CA	client
Certificate Renewal Request (sec. 3.5.2)	certificate request containing new public key with proof of possession and current certificate serial number; signed with current private key	certificate holder	Issuer CA
	CA returns signed certificate and CA's certificate or an error message	Issuer CA	certificate holder

new certificate. If the request was rejected, the message contains error codes. This transaction is described in detail in section 3.5.3.[14]

2.4 Client Specifications

PKI Clients use the PKI to provide certificate processing functions for certificate holders and certificate users, including CAs and other end entities. End entities may also include ORAs, persons and computing systems (e.g., routers and firewalls).

At a minimum, PKI Clients validate signatures, obtain certificates and CRLs, and validate certification paths. PKI Clients that serve certificate holders also generate signatures and may support PKI transactions to revoke or renew their certificates.

2.4.1 Interoperability-Relevant PKI Client Functional Specifications

At a minimum, clients shall be able to:

- verify signatures;

[14] An alternative syntax for this transaction is presented in section 3.5.4.

- obtain certificates and CRLs from a repository; and
- validate certification paths.

2.4.2 PKI Client Transaction Set

Table 2-4 gives the summary of transactions used by clients. These transactions enable clients to obtain certificates and CRLs from the repository. All client transactions are performed with the certificate repository. All clients shall support the following transactions:

- Retrieve certificates - this transaction permits a user to bind to the directory service or a specified repository using LDAP and retrieve one or more certificate(s) according to:
  - subject name; or
  - certificate serial number and issuer's name.
- Retrieve a CRL - This transaction permits a user to bind to the directory service or a specified repository using LDAP and retrieve the current CRL for a particular CA, or a specifically identified CRL.

At a minimum, retrieval of certificates and CRLs using the Lightweight Directory Access Protocol (LDAP) shall be supported by all compliant clients. These transactions are described further in [RFC1777].

Table 2-4 Client Electronic Transaction Set

Transaction	Description	From	To
Retrieve Certificate (see sec. 3.5.6)	Query repository or specified repository for an entity's certificate(s)	client	repository
	return certificate or error message to requester	repository	client
Retrieve CRL (sec. 3.5.7)	Query repository or specified repository for latest CRL issued by a particular CA	client	repository
	return CRL to requester	repository	client

3. Data Formats

Basic data formats must be defined for interoperability of PKI components. The data formats include certificate, CRL, and transaction formats. These specifications include data formats for all transactions between infrastructure components, and between PKI clients and infrastructure components.

3.1 Certificate Format

The X.509 V3 certificate format shall be used. Although the revision to ITU-T Recommendation X.509 that specifies the version 3 format is not yet published, the version 3 format has been widely adopted and is specified in American National Standards Institute X9.55-1995 [X9.55], and the Internet Engineering Task Force's Internet Public Key Infrastructure working document [PKIX1]. The X.509 version 3 certificate includes the following:

Version
Serial Number
Issuer Signature Algorithm
Issuer Distinguished Name
Validity Period
Subject Distinguished Name
Subject Public Key Information
Issuer Unique Identifier (optional)
Subject Unique Identifier (optional)
Extensions (optional)
Issuer's Signature on all the above fields

3.1.1 Certificate Fields

The Abstract Syntax Notation One (ASN.1) definition of the X.509 certificate syntax is stated in Appendix A. For signature calculation, the certificate is encoded under the ASN.1 Distinguished Encoding Rules (DER). ASN.1 DER encoding is a tag, length, value encoding system for each element.[ISO25-1]

The following items specify the use of the X.509 v3 certificate. With the exception of the optional **subjectUniqueID** and the **issuerUniqueID** fields, CAs shall generate these fields and clients shall be capable of processing them in accordance with the X.509 standard. CAs shall not issue certificates containing the optional **subjectUniqueID** and the **issuerUniqueID** fields. Clients are not required to process **subjectUniqueID** and the **issuerUniqueID** fields; however, they shall reject certificates containing these fields if they do not process them.

Version

The **version** field describes the version of the encoded certificate. The value of this field shall be 2, signifying a version 3 certificate.

Serial number

The **serialNumber** is an integer assigned by the CA to each certificate. It shall be unique for each certificate issued by a given CA (i.e., the issuer name and serial number identify a unique certificate).

Signature

The **signature** field contains the algorithm identifier for the algorithm used to sign the certificate. The **signature** field includes an **algorithmIdentifier**, which, in principle may be used to pass parameters. Certificates conforming to this interoperability specification shall be signed with either the DSS, RSA or ECDSA algorithms, and the contents of the **algorithmIdentifier** field shall be as specified in section 3.1.2.1. Certificates shall not use the **signature** field to pass parameters (see Subject Public Key Information below) since this field is not protected by the issuer's signature.[15]

Issuer Name

The **issuer** field provides a globally unique identifier of the authority signing the certificate. The syntax of the issuer name is an X.500 distinguished name. The distinguished name is composed of AttributeType - AttributeValue pairs. In general, the AttributeType will be defined by the X.500 series of recommendations; AttributeValue will be of type DirectoryString.

DirectoryString is a choice of PrintableString, TeletexString, and UniversalString. PrintableString is a basic Latin character set supporting upper and lowercase letters, digits, and a handful of special characters. TeletexString is a superset of PrintableString, adding Latin characters with accents and Japanese characters. UniversalString is a multi-octet character set including all the major character sets.

Conforming CAs shall always use the most restrictive choice when constructing a **DirectoryString**. That is, an **AttributeValue** which requires only basic Latin characters shall always be represented as **PrintableString**. An **AttributeValue** that includes accented Latin characters shall be represented as **TeletexString**. **UniversalString** shall only be used if the character set for **TeletexString** is insufficient.

Alternative names may be supplied in the **issuerAltName** extension and some users of X.509 certificates apparently contemplate a null **issuer** field. However, certificates conforming to this interoperability specification shall contain the X.500 distinguished name of the certificate issuer in this field.

Validity

The **validity** field indicates the dates on which the certificate becomes valid (**notBefore**) and on which the certificate ceases to be valid (**notAfter**). The validity field may represent dates in **UTCTime** or **GeneralizedTime**. For this specification, the validity field shall always use **UTCTime**.

The **UTCTime** (Coordinated Universal Time) values included in this field shall be expressed in Greenwich Mean Time (Zulu) and shall express granularity to the second. Seconds shall be

[15] See "A Security Flaw in the X.509 Standard," available from http://www.cygnacom.com/docfiles/dsaflaw.zip, for the rationale for excluding parameters from this field.

explicitly stated, even if zero. **UTCTime** shall be expressed as YYMMDDHHMMSSZ. The year field shall be interpreted as follows:

- if YY is equal to or greater than 50, the year shall be 19YY; and
- if YY is less than 50, the year shall be 20YY.

<u>Subject Name</u>

The purpose of the **subject** field is to provide a unique identifier of the subject of the certificate. The syntax of the subject name shall be an X.500 distinguished name. As described for issuer names, conforming CAs shall use the most restrictive choice when constructing **DirectoryString**s. Alternative names may be supplied in the **subjectAltName** extension and some users of X.509 certificates apparently contemplate a null **subject** field. However, certificates conforming to this interoperability specification shall contain the subject's X.500 distinguished name in this field.

<u>Subject Public Key Information</u>

The **subjectPublicKeyInfo** field is used to carry the public key and identify the algorithm with which the key is used. It includes the **subjectPublicKey** field and an **algorithmIdentifier** field with **algorithm** and **parameters** subfields. Certificates conforming to this interoperability specification shall use either the DSS, RSA or ECDSA algorithms, and the contents of the **algorithmIdentifier** field shall be as specified in section 3.1.2.1. The **parameters** subfield of the **subjectPublicKeyInfo** field shall be the only method used to pass or obtain DSS or ECDSA parameters.

<u>Unique Identifiers</u>

The **subjectUniqueIdentifier** and **issuerUniqueIdentifier** fields are present in the certificate to handle the possibility of reuse of subject and/or issuer names over time. Compliant CAs shall not issue certificates that include these unique identifiers. Compliant PKI clients are not required to process certificates that include these unique identifiers. However, if they do not process these fields, they are required to reject certificates that include these fields.

<u>Extension</u>

The addition of the **extension** field is the principal change introduced to X.509 v3 certificates. Extensions have three components: **extnId**, that names the extension, **critical,** the criticality flag that specifies that the extension is critical or noncritical, and **extnValue,** the extension value. A certificate may contain any number of extensions, including locally defined extensions. If the criticality flag is set, a client shall either be able to process that extension, or shall not validate the certificate.

A set of standardized extensions has been developed in an amendment to the X.509 standard [DAM]. The use of these standardized extensions in conforming implementations is specified in section 3.1.3 below.

<u>Issuer's Signature</u>

The actual signature on the certificate is defined by the use of the **SIGNED** parameterized type, which expands to a **SEQUENCE** of the data being signed (i.e., the certificate), an algorithm identifier, and a **BIT STRING** which is the actual signature. The **algorithmIdentifier** that identifies the algorithm used to sign the certificate. Although this **algorithimIdentifier** field includes a **parameters** field that can, in principle, be used to pass the parameters used by the signature algorithm (see sec. 3.1.2.1), it is not itself a signed object. The **parameters** field of the certificate signature shall not be used to pass parameters. When parameters are used to validate a signature, they shall be obtained from the **subjectPublicKeyInfo** field of the issuing CA's certificate.

3.1.2 Cryptographic Algorithms

This document specifies two classes of cryptographic algorithms; digital signature algorithms and message authentication algorithms. Digital signature algorithms are always identified with a secure hash algorithm.

At a minimum, a conforming PKI component shall implement one of the identified digital signature algorithms.

3.1.2.1 Digital Signature Algorithms

X.509 certificates specify both the algorithm used to sign the certificate (in the **signature** field) and the algorithm of the subject's public key (in the **subjectPublicKeyInfo** field). The two algorithms may be different. CAs shall be able to sign certificates and Certificate Revocation Lists (CRLs) using at least one of the three algorithms as specified below. End entities shall be able to sign with at least one of the three algorithms listed below. Clients shall be able to validate signatures of at least one of the types specified below. To achieve maximum interoperability, it is recommended that clients be capable of validating signatures for all three of the algorithms specified below.

<u>RSA</u>

The RSA signature algorithm is defined in PKCS #1 [PKCS#1]. Although RSA can be used with several hash algorithms, the only variant used to sign certificates and CRLs conforming to this interoperability specification is RSA with the SHA-1 hash algorithm specified in FIPS 180-1 [FIPS 180]. For this specification, the following ASN.1 object identifier is used to identify RSA with SHA-1:

```
sha-1WithRSAEncryption OBJECT IDENTIFIER  ::=  {
   iso(1) identified-organization(3) oiw(14)
   secsig(3) algorithm(2) 29  }
```

This object identifier shall appear in the parameterized type **SIGNED** and the **signature** field in both certificate or CRL signed with RSA. Whenever this object identifier appears as the value for **algorithmIdentifier**, the parameters component shall be **NULL**.

When a certificate or CRL is signed with RSA and SHA-1, the signature shall be generated and encoded as follows:

> The certificate or CRL is ASN.1 DER encoded, and is used as the input to the SHA-1 hash function. The SHA-1 output value is ASN.1 encoded as an **OCTET STRING** and the result is encrypted with the RSA algorithms to form the signed quantity. When signing, the RSA algorithm generates an integer y. This signature value is then ASN.1 encoded as a

BIT STRING, such that the most significant bit in y is the first bit in the bit string and the least significant bit in y is the last bit in the bit string, and included in the **Certificate** or **CertificateList** (in the **signature** field).

(In general the conversion to a bit string occurs in two steps. The integer y is converted to an octet string such that the first octet has the most significance and the last octet has the least significance. The octet string is converted into a bit string such that the most significant bit of the first octet shall become the first bit in the bit string, and the least significant bit of the last octet is the last bit in the **BIT STRING**.)

When a conforming CA issues a certificate whose **subjectPublicKeyInfo** field contains an RSA public key, the object identifier **rsaEncryption** shall appear as the **algorithmIdentifier** in the **subjectPublickeyInfo** field to identify the key as an RSA public key.

```
pkcs-1 OBJECT IDENTIFIER ::= { iso(1) member-body(2) US(840)
        rsadsi(113549) pkcs(1) 1 }

rsaEncryption OBJECT IDENTIFIER ::=  { pkcs-1 1}
```

Whenever the **rsaEncryption** object identifier is used in the algorithm field of a value of type **AlgorithmIdentifier**, the parameters field shall have ASN.1 type **NULL**.

The rsa public key shall be encoded using the ASN.1 type **RSAPublicKey**:

```
RSAPublicKey ::= SEQUENCE {
  modulus            INTEGER, -- n
  publicExponent     INTEGER  -- e
              }
```

where modulus is the modulus n, and **publicExponent** is the public exponent e. The DER encoded **RSAPublicKey** is the value of the **BIT STRING subjectPublicKey**.

This object identifier is used in public key certificates for both RSA signature keys and RSA encryption keys. The intended application for the key may be indicated in the key usage field (see sec. 4.2.1.3). The use of a single key for both signature and encryption purposes is not recommended, but is not forbidden.

<u>DSS</u>

The Digital Signature Algorithm is defined in FIPS 186 [FIPS186]. The ASN.1 object identifier used to identify DSS public keys shall be:

```
id-dsa ID ::= { iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) dsa(1) }
```

The Digital Signature Standard (DSS) [FIPS186] specifies that DSA shall be used with the SHA-1 hash algorithm. The ASN.1 object identifier used to identify DSS signatures shall be:

```
id-dsa-with-sha1 ID ::= {
        iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) id-dsa-with-sha1(3) }
```

The **AlgorithmIdentifier** within **subjectPublicKeyInfo** is the only place within a certificate where id-dsa shall be used. The id-dsa algorithm syntax includes optional parameters. These parameters are commonly referred to as p, q, and g. If the DSA algorithm parameters are absent from the **subjectPublicKeyInfo AlgorithmIdentifier** and the CA signed the subject certificate using DSA, then the certificate issuer's DSA parameters apply to the subject's DSA key. If the DSA algorithm parameters are absent from the **subjectPublicKeyInfo AlgorithmIdentifier** and the CA signed the certificate using a signature algorithm other than DSA, then clients shall not validate the certificate. The parameters are included using the following ASN.1 structure:

```
DSAParameters  ::=  SEQUENCE  {
        prime1          INTEGER, -- modulus p
        prime2          INTEGER, -- modulus q
        base            INTEGER  } – base g
```

The **id-dsa-with-sha1** algorithm identifier shall be used in the **SIGNED** parameterized type (e.g., in the signature on a certificate or CRL) and the **signature** fields of certificates and CRLs. The **id-dsa-with-sha1** algorithm syntax has NULL parameters. The DSA parameters in the certificate of the issuer shall apply to the verification of the signature.

The DSA public key shall be ASN.1 encoded as an **INTEGER**; this encoding shall be used as the contents (i.e., the value) of the **subjectPublicKey** component (a **BIT STRING**) of the **SubjectPublicKeyInfo** data element.

```
DSAPublicKey ::= INTEGER -- public key Y
```

When signing, the DSA algorithm generates two values. These values are commonly referred to as r and s. To easily transfer these two values as one signature, they shall be ASN.1 encoded using the following ASN.1 structure:

```
Dss-Sig-Value  ::=  SEQUENCE  {
        r       INTEGER,
        s       INTEGER  }
```

The encoded signature is conveyed as the value of the **BIT STRING** in the **SIGNED** parameterized type in a certificate or CertificateList.

<u>ECDSA</u>

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in the draft ANSI X9.62 standard [X9.62]. The ASN.1 object identifier used to identify the ECDSA algorithm shall be:

```
ansi-X9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10045 }
```

When used to sign certificates, CRLs, or PKI messages, the ECDSA shall be used with the SHA-1 hash algorithm. The ASN.1 object identifier used to identify the ECDSA algorithm with SHA-1 shall be:

```
ecdsa-with-SHA1 OBJECT IDENTIFIER  ::=  { ansi-X9-62 1 }
```

When the **ecdsa-with-SHA1** algorithm identifier is used in the SIGNED parameterized TYPE (e.g., in the signature on a certificate or CRL) it shall have NULL parameters. The ECDSA parameters in the certificate of the issuer shall apply to the verification of the signature.

When signing, the ECDSA algorithm generates two values. These values are commonly referred to as r and s. To easily transfer these two values as one signature, they shall be ASN.1 encoded using the following ASN.1 structure:

```
Ecdsa-Sig-Value  ::=  SEQUENCE  {
```

```
r       INTEGER,
s       INTEGER  }
```

When certificates contain an ECDSA public key, the **id-ecPublicKey** algorithm identifier shall be used. The **id-ecPublicKey** algorithm identifier is defined as follows:

```
id-public-key-type OBJECT IDENTIFIER  ::= { ansi-X9.62 2 }

id-ecPublicKey OBJECT IDENTIFIER ::= { id-publicKeyType 1 }
```

The elliptic curve public key (an **ECPoint** which is an **OCTET STRING**) is mapped to a **subjectPublicKey** (a **BIT STRING**) as follows: the most significant bit of the **OCTET STRING** becomes the most significant bit of the **BIT STRING**, etc.; the least significant bit of the **OCTET STRING** becomes the least significant bit of the **BIT STRING**.

ECDSA requires use of certain parameters with the public key. The parameters may be included in the certificate using the following ASN.1 structure:

```
ECParameters  ::=  SEQUENCE  {
        version                 INTEGER    { ecpVer1(1) }    (ecpVer1),
                                        -- version is always 1
        fieldID         FieldID      { {FieldTypes} },
                                        -- identifies the finite field over
                                        -- which the curve is defined
        curve           Curve,          -- coefficients a and b of the elliptic curve
        base            ECPoint,        -- specifies the base point P
                                        -- on the elliptic curve
        order           INTEGER,        -- the order n of the base point
        cofactor        INTEGER,
        ...
        }


FieldElement ::= OCTET STRING
Curve ::= SEQUENCE {
        a       FieldElement,
        b       FieldElement,
        seed    BIT STRING OPTIONAL
        }

ECPoint ::= OCTET STRING
```

The components of type **ECParameters** have the following meanings:

- **version** specifies the version number of the elliptic curve parameters. It shall have the value 1 for this version of the Standard. The notation above creates an **INTEGER** named **ecpVer1** and gives it a value of one. It is used to constrain **version** to a single value.
- **fieldID** identifies the finite field over which the elliptic curve is defined. Finite fields are represented by values of the parameterized type **FieldID**, constrained to the values of the objects defined in the information object set **FieldTypes**. Additional detail regarding **fieldID** is provided below.

- **curve** specifies the coefficients a and b of the elliptic curve E. Each coefficient shall be represented as a value of type **FieldElement**, an **OCTET STRING. seed** is an optional parameter used to derive the coefficients of a randomly generated elliptic curve.
- **base** specifies the base point P on the elliptic curve. The base point shall be represented as a value of type **ECPoint**, an **OCTET STRING**.
- **order** specifies the order n of the base point.
- **cofactor** is the integer $h = \#E(F_q)/n$.

The **AlgorithmIdentifier** within **subjectPublicKeyInfo** is the only place within a certificate where the parameters may be used. If the ECDSA algorithm parameters are absent from the **subjectPublicKeyInfo AlgorithmIdentifier** and the CA signed the subject certificate using ECDSA, then the certificate issuer's ECDSA parameters apply to the subject's ECDSA key. If the ECDSA algorithm parameters are absent from the **subjectPublicKeyInfo AlgorithmIdentifier** and the CA signed the certificate using a signature algorithm other than ECDSA, then clients shall not validate the certificate.

```
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {
  fieldType   FIELD-ID.&id({IOSet}),
  parameters  FIELD-ID.&Type({IOSet}{@fieldType})  OPTIONAL
}

FieldTypes FIELD-ID ::= {
  { Prime-p            IDENTIFIED BY prime-field } |
  { Characteristic-two IDENTIFIED BY characteristic-two-field },
  ...
}

FIELD-ID ::= TYPE-IDENTIFIER
```

FieldID is a parameterized type composed of two components, **fieldType** and **parameters**. These components are specified by the fields **&id** and **&Type**, which form a template for defining sets of information objects, instances of the class **FIELD-ID**. This class is based on the useful information object **class TYPE-IDENTIFIER**, described in X.681 Annex A. In an instance **of FieldID, "fieldType"** will contain an object identifier value that uniquely identifies the type contained in **"parameters."** The effect of referencing **"fieldType"** in both components of the **fieldID** sequence is to tightly bind the object identifier and its type.

The information object set **FieldTypes** is used as the single parameter in a reference to type **FieldID**. **FieldTypes** contains two objects followed by the extension marker ("..."). Each object, which represents a finite field, contains a unique object identifier and its associated type. The values of these objects define all of the valid values that may appear in an instance of **fieldID**. The extension marker allows backward compatibility with future versions of this standard which may define objects to represent additional kinds of finite fields.

The object identifier **id-fieldType** represents the root of a tree containing the object identifiers of each field type. It has the following value:

```
id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1) }
```

The object identifiers **prime-field** and **characteristic-two-field** name the two kinds of fields defined in this Standard. They have the following values:

```
prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }
characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }

Prime-p ::= INTEGER    -- Field size p

Characteristic-two ::= SEQUENCE {
  m              INTEGER,                -- Field size 2^m
  basis          CHARACTERISTIC-TWO.&id({BasisTypes}),
  parameters     CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}

BasisTypes CHARACTERISTIC-TWO::= {
  { NULL                IDENTIFIED BY onBasis } |
  { Trinomial           IDENTIFIED BY tpBasis } |
  { Pentanomial         IDENTIFIED BY ppBasis },
  ...
}

Trinomial ::= INTEGER
Pentanomial ::= SEQUENCE {
  k1  INTEGER,
  k2  INTEGER,
  k3  INTEGER
}

CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
```

The object identifier **id-characteristic-two-basis** represents the root of a tree containing the object identifiers for each type of basis for the characteristic-**two finite fields. It has the following value:**

```
id-characteristic-two-basis OBJECT IDENTIFIER ::= {
                  characteristic-two-field basisType(1) }
```

The object identifiers **onBasis**, **tpBasis** and **ppBasis** name the three kinds of basis for characteristic-two finite fields defined by [X9.62]. They have the following values:

```
onBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis  OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
```

3.1.2.2 Message Authentication Algorithms

The following message authentication algorithm OIDS are recognized:

```
DES-MAC OBJECT IDENTIFIER ::= {
  iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 10
              -- carries length in bits of the MAC as
              -- an INTEGER parameter, constrained to 32
              -- for this specification
  }
```

This algorithm provides integrity by computing a DES MAC (as specified by [FIPS-113]) on data. The length of the MAC shall be 32 bits for this specification.

3.1.3 Certificate Extensions

A set of standardized extensions has been developed and is specified in an amendment to X.509 [DAM]. Extensions have three components: extension name, criticality flag, and extension value. As specified in the amendment to X.509 [DAM], clients shall not validate certificates that contain an extension with the criticality flag set, unless the client can process that extension.

The standardized extensions that have been defined may be divided into four categories: key and policy information; subject and issuer attributes; certification path constraints; and CRL identification extensions.

3.1.3.1 Key and Policy Information

These extensions provide information to identify a particular public key and certificate. They can be used to identify a particular public key/certificate for a CA which has several certificates. This may help a client to find the particular CA certificate needed to establish a certification path. These extensions may restrict the purposes for which a key may be used, and provide information in CA certificates about equivalent policies.

Authority Key Identifier

The **authorityKeyIdentifier** extension provides a means of identifying the particular private key used to sign a certificate. The identification can be based on either the key identifier or on the issuer name and serial number. The key identifier method shall be used in certificates conforming to this interoperability specification. This extension is used where an issuer has multiple signing keys (either due to multiple concurrent key pairs or due to changeover). CAs shall be capable of generating this extension, and clients shall be capable of finding and validating certification paths where the issuing CA has several digital signature keys. It is recommended that clients be able to process either the key identifier or the certificate issuer plus certificate serial number form of key identifier to help find certification paths.

Subject Key Identifier

This field enables differentiation of keys held by a subject. This field shall be included in every certificate issued. This extension shall be noncritical.

Key Usage

The **keyUsage** extension defines restrictions on the use of the key contained in the certificate based on policy and/or usage (e.g., signature, encryption). CAs shall support the generation of this extension and clients shall be capable of processing it. While **KeyUsage** is defined as a **BIT STRING**, conforming CAs shall set only one value within this string in end-entity certificates. For example, **KeyUsage** shall not be both **digitalSignature** and **dataEncipherment** in an end-entity certificate. This extension shall be set to critical.

Private Key Usage Period

The **privateKeyUsagePeriod** extension applies only to digital signature keys. A signature on a document that purports to be dated outside the private key usage period is not valid.[16] CAs may generate certificates containing this extension but conforming clients are not required to process it.

Extended Key Usage

The extendedKeyUsage extension defines application-specific restrictions on the use of keys contained in a certificate. When this extension is used, interoperability is not a factor. Conforming PKI components are not required to support this extension.

Certificate Policies

The **certificatePolicies** extension contains one or more object identifiers (OIDs). Each OID indicates a policy under which the certificate has been issued. CAs shall be able to generate certificates with one or more instances of **policyIdentifier**.

Clients shall be capable of processing **policyIdentifier** fields against a list of acceptable policies. (The list of policies is dependent upon on application requirements.) Clients shall compare the policy identifier(s) in the certificate to that list. Clients shall validate the certification path only if at least one of the policy OIDs in the **certificatePolicies** field in each certificate in the path matches one of the policies in the list of acceptable policies.

Conforming components are required to process the **policyQualifiers** subfield of **certificatePolicies** if present, and shall support the policy qualifiers **id-pkix-cps** and **id-pkix-unotice** (see [PKIX1].) Conforming CAs need not be able to generate this subfield.

Policy Mapping

This noncritical extension is used in CA certificates. It lists pairs of object identifiers; each pair includes an **issuerDomainPolicy** and a **subjectDomainPolicy**. The pairing indicates that the issuing CA considers its **issuerDomainPolicy** equivalent to the subject CA's **subjectDomainPolicy**. CAs shall be capable of generating the **policyMappings** extension. Clients shall be capable of processing this extension.

---

[16] Note that verification of time associated with a signature implies use of a notary or trusted timestamp. Both are outside the scope of this specification.

3.1.3.2 Certificate Subject and Issuer Attributes

The **subjectAltName**, **issuerAltName** and **subjectDirectoryAttributes** are all noncritical extensions. They provide additional information about other names and attributes of the subject and issuer.

Alternative Name

The **subjectAltName** and **issuerAltName** extensions allow additional identities to be bound to the subject and issuer of the certificate. Defined options include an RFC822 [RFC 822] name (electronic mail address), a DNS name, and a uniform resource identifier (URI.) Multiple instances may be included. Whenever such identities are to be bound in a certificate, the **subjectAltName** or **issuerAltName** fields shall be used.[17]

The **subjectAltName** and **issuerAltName** extensions are normally noncritical in certificates conforming to this interoperability specification. An implementation which recognizes these extensions need not be able to process all the alternatives of the choice. If the alternative used is not supported by the implementation, the extension field is ignored.

This specification defines the semantics with associated with an **issuerAltName** field containing a URI. The URI specifies the location of the issuer's certificate(s) which contain the public key material corresponding to the private key used to sign the certificate. The semantics associated with other classes of identities, or any **subjectAltName** entries, are not defined in this specification.

If a CA's certificates are not available from a well-known X.500 directory service, the CA shall include URI alternative names specifying the location of the issuer's certificate(s). Clients are required to process the URI alternative name format and must recognize the LDAP URL [RFC1959]. Clients are not required to recognize any other URI formats.

Subject Directory Attributes

The **subjectDirectoryAttributes** extension may hold any information about the subject where that information has a defined X.500 Directory attribute. This extension is always noncritical. Implementation and use of this extension is optional.

3.1.3.3 Certification Path Constraints

The **basicConstraints**, **nameConstraints** and **policyConstraints** all apply restrictions to valid certification paths.

Basic Constraints

The **basicConstraints** extension tells whether the subject of the certificate is a CA through the **cA** component and the lengths of certification paths through the **pathLenConstraint** component. CAs shall support the generation of the **basicConstraints** extension in certificates and clients shall be

[17] X.509 allows null certificate **subject** or **issuer** field accompanied by a critical **subjectAltName** or **issuerAltName** giving the name in an alternative format. Such certificates are not supported by this interoperability specification.

capable of processing it. The **pathLenConstraint** component is meaningful only if **cA** is set to TRUE.

The **basicConstraints** extension shall be included in all certificates. End entity certificates shall contain a **basicConstraints** extension with an empty SEQUENCE value. CA certificates shall contain a **basicConstraints** extension the **cA** component set to TRUE. The **basicConstraints** extension shall be marked as critical in all certificates issued to CAs.

<u>Name Constraints</u>

The **nameConstraints** field applies only to CA certificates. It indicates a name space in which all subsequent certificates in a certification path must be located. CAs shall be capable of including this field in certificates and clients shall be capable of processing it. If used, it shall be critical.

<u>Policy Constraints</u>

The **policyConstraints** extension serves two functions. It can require that a specific policy apply to all or to a portion of the CA path. It can also inhibit policy mapping for all or a selected portion of the certification path. CAs shall be capable of supporting the issuance of certificates with this extension, and clients shall be capable of processing this extension. If used, it shall be critical.

3.1.3.4 CRL Identification Extensions

These extensions include information in a certificate about where to obtain the Certificate Revocation List (CRL) that applies to that certificate. They facilitate the division of a CA's potentially large CRL into several shorter CRLs, by identifying in the certificate which CRL applies to a certificate and give the name of the CRL issuer (which may be a CA other than the CA that issued the certificate).

<u>CRL Distribution Points</u>

The **cRLDistributionPoints** extension identifies the CRL distribution point or points to which clients should refer to ascertain if a certificate has been revoked. This field has three component fields: **distributionPoint**, **reasons** and **cRLIssuer**.

- The **distributionPoint** component identifies the location from which the CRL can be obtained. If this field is absent, the CRL distribution point name defaults to the issuer name. This extension provides a mechanism to divide the CRL into manageable pieces if the CA has a large constituency.
- The **reasons** component identifies the reasons for revocation covered by the CRL issued by the corresponding **distributionPoint**. If the **reasons** component is absent, the corresponding **distributionPoint** distributes a CRL which will contain an entry for this certificate, if it has been revoked for any reason. Clients are not required to process the **reasons** component.
- The **cRLIssuer** component identifies the authority that issues and signs the CRL. If this component is absent, the CRL issuer name defaults to the certificate issuer name. One use for this component is to allow the construction of consolidated CRLs, that include certificates issued by more than one CA.

CAs shall include the **cRLDistributionPoints** extension with a **distributionPoint** component. If a CA's CRLs are not available from a well-known X.500 directory service, the CA shall include URI alternative names specifying the location of the current CRL for this certificate in the **distributionPoint** component. Clients shall be able process the **cRLDistributionPoints** extension; they must recognize the URI format and process at a minimum the LDAP URI. Clients shall be able to use distribution point CRLs and validate CRLs where the **cRLIssuer** component is used. See section 3.2.2 below for a further discussion of distribution points.

Table 3-1 Summary of Standardized Certificate Extensions

Extension	Used By	Use	Critical
Key and Policy Information			
keyIdentifier	all	identifies the key used to sign this certificate (the signing CA may have several keys)	No
authorityKeyIdentifier	all	unique with respect to authority.	
authorityCertIssuer	all	identifies issuing authority of CA's certificate; alternative to key identifier	
authorityCertSerialNumber	all	used with authorityCertIssuer	
subjectKeyIdentifier	all	enables differentiation of different keys for same subject. Must be unique for subject.	No
keyUsage	all	defines allowed purposes for use of key (e.g., digital signature, key agreement...)	Yes*
extendedKeyUsage	all	defines application-specific purposes for keys	No*
privateKeyUsagePeriod	all	digital signature keys only. Signatures on documents that purport to be dated outside the period are invalid.	No*
certificatePolicies	all	policy identifiers and qualifiers that identify and qualify policies applying to the certificate	No*
policyIdentifiers	all	the OID of a policy.	
policyQualifiers	all	more information about the policy	
policyMappings	CA	indicates equivalent policies	No
Certificate Subject and Issuer Attributes			
subjectAltName	all	used to list alternative names (e.g., rfc822 name, X.400 address, IP address...)	No*
issuerAltName	all	used to list alternative names	No*
subjectDirectoryAttributes	all	any attributes (e.g., supported algorithms)	No
Certification Path Constraints			
basicConstraints	all	constraints on subject's role & path lengths	Yes*
cA	all	distinguish CA from end entity cert.	
pathLenConstraint	CA	max. number of following CAs in cert. path; 0 indicates that CA only issues end entity certs.	
nameConstraints	CA	limits subsequent CA cert. Name space.	Yes*
permittedSubtrees	CA	names outside indicated subtrees are forbidden	
excludedSubtrees	CA	indicates disallowed subtrees	
policyConstraints	all	constrains certs. Issued by subsequent CAs	Yes*
requireExplicitPolicy	all	All certs. following in the cert. path must contain an acceptable policy identifier	
inhibitPolicyMapping	all	prevent policy mapping in following certs.	
CRL Identification			
crlDistributionPoints	all	divides long CRL into shorter lists	No*
distributionPoint	all	location from which CRL can be obtained	
reasons	all	reasons for cert. inclusion in CRL	
cRLIssuer	all	name of component that issues CRL.	

NOTES:

* Standard allows either critical or noncritical. Indication is for use in interoperable implementations.

Table 3-2 Use of Standardized Certificates by the MISPC

Extension	Certificate	Client
Key and Policy Information		
authorityKeyIdentifier		
authorityKeyIdentifier	to be included in all certs issued: a random number large enough to generally be globally unique	optional - may be used to help find cert. paths where issuer has multiple certs. (1)
authorityCertIssuer	not used	optional - used to find cert. paths where issuer has multiple certs. (1)
authorityCertSerialNumber	not used	
subjectKeyIdentifier	to be included in all certs issued: a random number large enough to generally be globally unique	optional: used with CRLs to identify revoked certificates.
keyUsage	supported	supported
extendedKeyUsage	not used	not used
privateKeyUsagePeriod	supported	optional
certificatePolicies		
policyIdentifiers	supported	supported; compared during cert. path validation with a list of acceptable policies
policyQualifiers	used only in CA certificates	supported (see 3.1.3.1)
policyMappings	supported	supported
Certificate Subject and Issuer Attributes		
subjectAltName	supported	not used
issuerAltName	supported	not used
subjectDirectoryAttributes	not used	not used
Certification Path Constraints		
basicConstraints		
cA	used in all certificates	supported
pathLenConstraint	supported	supported
nameConstraints		
permittedSubtrees	supported	supported
excludedSubtrees	supported	supported
policyConstraints		
requireExplicitPolicy	supported	supported
inhibitPolicyMapping	supported	supported
CRL Identification		
cRLDistributionPoints		
distributionPoint	supported	supported
reasons	supported	supported
cRLIssuer	supported	supported

NOTES:

For Certificates, “supported” means that CAs shall be able to issue certificates that contain this extension. For clients, “supported” means that the client shall be capable of processing this extension.

(1) Clients shall be capable of finding certification paths where CAs have multiple certificates, whether or not they use this extension to do so.

3.1.3.5 Summary of Certificate Extension Use

Table 3-1 summarizes the standardized certificate extensions, while Table 3-2 summarizes the use by the MISPC of standardized extensions for certificates and clients.

3.2 Certificate Revocation List (CRL)

Certificate Revocation Lists (CRL) are used to list unexpired certificates that have been revoked or placed on "hold." Certificates may be revoked for a variety of reasons, ranging from routine administrative revocations, (when the certificate's subject leaves the issuing organization, or when responsibilities and certificate attributes change), to situations where the private key is compromised. A "hold" indicates the CA will not vouch for the binding of the certificate subject and public key at this time.

The X.509 v2 certificate revocation list format is augmented by several optional extensions, similar in concept to those defined for certificates. CAs shall be able to generate X.509 v2 CRLs as specified below, and clients shall be capable of processing them when validating certification paths. The CA that issues a CRL is not necessarily the CA that issued the revoked certificate, and some CAs may issue only CRLs. The X.509 v2 CRL includes the following:

Version
Issuer Signature Algorithm
Issuer Distinguished Name
This Update
Next Update
Revoked Certificates, a sequence of zero or more of the following sequence:
    Certificate Serial Number
    Revocation Date
    CRL Entry Extensions (optional)
CRL Extensions (optional)
Issuer's Signature on all the above listed fields

3.2.1 CRL Fields

The X.509 v2 CRL ASN.1 syntax is given in Appendix B. For signature calculation, the data that is to be signed is ASN.1 DER encoded. ASN.1 DER encoding is a tag, length, value encoding system for each element.

The following items describe the use of the X.509 v2 CRL.

<u>Version</u>

This field describes the version of the encoded CRL. The value of this field shall be 1, indicating a v2 CRL.

<u>Signature</u>

The **signature** field contains the algorithm identifier for the algorithm used to sign the CRL. The contents are identical to the contents of the certificate **signature** field. Refer to <u>Signature</u> in section 3.1.1 for information about this field. The CRL may be signed with any of the algorithms identified in section 3.1.2.1; in general, the CA should sign the CRL with the same algorithm used to sign the certificates. Refer to section 3.1.2.1 for the signature algorithm object identifiers. The **parameters** subfield of the CRL **signature** field shall not be used to pass DSS parameters; rather DSS parameters shall be obtained from the **subjectPublicKeyInfo** field of the certificate of the issuing CA.

Issuer Name

The **issuer** field provides a globally unique identifier of the CA signing the CRL. The issuer name is an X.500 distinguished name. CRL issuer names with empty sequences are not supported by implementations conforming to this interoperability specification.

This Update

The **thisUpdate** field indicates the date of the CRL. This field may be represented as **UTCTime** or **GeneralizedTime**. For this specification, **thisUpdate** shall always be represented as **UTCTime** (Coordinated Universal Time) and shall follow the rules for the certificate **validity** field (see sec. 3.1.1 above).

Next Update

The **nextUpdate** field indicates the date by which the next CRL will be issued. The next CRL could be issued before the indicated date, but it will not be issued any later than the indicated date. This field may be represented as **UTCTime** or **GeneralizedTime**. For this specification, **nextUpdate** shall always be represented as **UTCTime** (Coordinated Universal Time) and shall follow the rules for the certificate **validity** field (see sec. 3.1.1 above).

Revoked Certificates

The **revokedCertificates** field is a list of the certificates that have been revoked. Each revoked certificate listed contains:

- the certificate serial number, stated in the **userCertificate** field. This element contains the value of **serialNumber** of the revoked certificate. This must be used in conjunction with the name of the issuing CA to identify an unexpired certificate that has been revoked.
- the **revocationDate** field that contains the date of the revocation in **UTCTime** format. The **UTCTime** (Coordinated Universal Time) value included in this field shall follow the rules for the certificate **validity** field (see sec. 3.1.1 above).
- optional CRL entry extensions, that are specified in section 3.2.3 below. The CRL entry extensions may give the reason that the certificate was revoked, state the date that the invalidity is believed to have occurred, and may state the name of the CA that issued the revoked certificate, which may be a different CA from the CA issuing the CRL. Note that the CA that issued the CRL is assumed to be the CA that issued the revoked certificate unless the **certificateIssuer** CRL entry extension is included.

3.2.2 CRL Extensions

The extensions defined by ISO/ITU for X.509 v2 CRLs provide methods for associating additional attributes with entire CRLs. Each CRL extension may be designated as critical or noncritical. A CRL validation shall fail if a client encounters a critical extension that it cannot process.

This section describes CRL extensions that shall be supported. A CRL extension is supported when: the CA is able to generate the extensions in a CRL and the clients are able to process the extension.

Authority Key Identifier

The **authorityKeyIdentifier** is a noncritical CRL extension that identifies the CA's key used to sign the CRL. This extension is useful when a CA uses more than one key; it allows distinct keys differentiated (e.g., as key updating occurs).   The identification can be based on either the key identifier or on the issuer name and serial number. The key identifier method shall be used, and the **keyIdentifier** shall be generated for all CRLs. This extension is useful where an issuer has multiple signing keys (either due to multiple concurrent key pairs or due to changeover). This extension shall be included in all CRLs, and clients shall be able to find and validate CRL certification paths where the issuing CA has multiple signing keys. Clients shall be able to process either the key identifier or the certificate issuer plus serial number form of **authorityKeyIdentifier** if they use this extension to find certification paths.

Issuer Alternative Name

The **issuerAltName** is a noncritical CRL extension that contains one or more alternative CA names. Whenever such alternative names are present in a CRL, they shall be placed in the issuer alternative name field. Implementations which recognize this extension need not be able to process all the alternative name formats. Unrecognized alternative name formats may be ignored by an implementation. CAs shall be capable of generating this extension in CRLs, however clients are not required to process it.

CRL Number

The **cRLNumber** field is a noncritical CRL extension which conveys a monotonically increasing sequence number for each CRL issued by a given CA through a specific CA directory entry or CRL distribution point. This extension can be used to alert certificate users to unscheduled issuance of full CRLs, or easily determine when a particular delete CRL supersedes another CRL. This extension shall be included in CRLs.

Issuing Distribution Point

The **issuingDistributionPoint** field is a critical CRL extension that identifies the CRL distribution point for this particular CRL. A distribution point is a directory entry that may be used to retrieve a CRL, and that may differ from the directory entry of the issuing CA. The CRL is signed by the CA's key. CRL distribution points do not have their own key pairs.

In addition, the **issuingDistributionPoint** field specifies CRLs that may contain only end entity certificates, or only CA certificates, or only certificates that have been revoked for a particular reason. Finally, this extension can identify an “indirect CRL,” that is a CRL that is issued by a different CA than the CA(s) that issued the revoked certificate. It contains the following components:

- **distributionPoint**, which gives the name of the distribution point name. If used, **distributionPoint** shall be an X.500 distinguished name;
- **onlyContainsUserCerts**, a Boolean value that indicates that the CRL contains only end entity certificates;
- **onlyContainsCACerts**, a Boolean value that indicates that the CRL contains only CA certificates;
- **onlySomeReasons**, a **ReasonsFlag** bit string that indicates the reasons for which certificates are listed in the CRL. Only the following reason flags shall be included in CRLs:
  - **keyCompromise** shall be used to indicate compromise or suspected compromise;
  - **cACompromise** shall be used to indicate that the certificate has been revoked because of a CA key compromise. It shall only be used to revoke CA certificates;
  - **affiliationChanged** shall be used to indicate that the certificate was revoked because of a change of affiliation of the certificate subject;
  - **superseded** shall be used to indicate that the certificate has been superseded;
  - **cessationOfOperation** shall be used to indicate that the certificate is no longer needed for the purpose for which it was issued, but there is no reason to suspect that the private key has been compromised.
- **indirectCRL**, a Boolean value that indicates that this is an indirect CRL.

Clients shall be able to process this field.

Delta CRL Indicator

The **deltaCRLIndicator** is a critical CRL extension that identifies a delta-CRL. The use of delta-CRLs can significantly improve processing time for applications which store revocation information in a format other than the CRL structure. This allows changes to be added to the local database while ignoring unchanged information that is already in the local database.

The value of **BaseCRLNumber** identifies the CRL number of the base CRL that was used as the starting point in the generation of this delta-CRL. The delta-CRL contains the changes between the base CRL and the current CRL. A delta-CRL is not issued by itself; if a delta-CRL is issued a complete current CRL is also issued. It is the decision of a CA as to whether to provide delta-CRLs. A delta-CRL shall not be issued without a corresponding base CRL. The value of CRL number for both the delta-CRL and the corresponding base CRL shall be identical.

A client constructing a locally held CRL from delta-CRLs shall consider the constructed CRL incomplete and unusable if the CRL number of the received delta-CRL is more that one greater that the CRL number of the delta-CRL last processed.[18] Support of delta-CRLs by clients and CAs is optional.

Summary of CRL Extension Use

Table 3-3 summarizes the standardized CRL extensions, while Table 3-4 summarizes the use of the standardized CRL extensions for the MISPC.

[18] Note that use of delta CRLs imposes an additional security requirement on clients; they must be capable of securely maintaining the composite CRL.

Table 3-3 Summary of CRL Extensions

Extension	Use	Critical
authorityKeyIdentifier	identifies the CA key used to sign CRL.	No
keyIdentifier	unique key identifier; alternative to **certIssuer** & **authorityCertSerialNumber**	
certIssuer	name of CA's cert. issuer	
authorityCertSerialNumber	used with **certIssuer**; combination must be unique	
issuerAltName	alternate name of CRL issuer	No*
cRLNumber	sequence number for CRL	No
issuingDistributionPoint	name of CRL distribution point; also gives reasons for revocations contained in CRL.	Yes
deltaCRLIndicator	indicates delta CRL (lists certificates. revoked since last full CRL) & gives sequence number	Yes

NOTES:

* Standard allows either critical or noncritical. Indication is for use in interoperable implementations.

3.2.3 CRL Entry Extensions

The CRL entry extensions defined for X.509 v2 CRLs provide methods for associating additional attributes with CRL entries. Each extension in a CRL entry is designated as critical or noncritical. A CRL validation shall fail if it encounters a critical CRL entry extension which it does not know how to process. However, an unrecognized noncritical CRL entry extension may be ignored.

Table 3-4 Summary of CRL Extensions and their use in the MISPC

Extension	CRL	Clients
authorityKeyIdentifier		
keyIdentifier	included in all CRLs issued	optional - used to help find correct CA certificate to validate CRL (1)
certIssuer	not generated	optional - issuer/serial number pair used to help find correct authority certificate to validate CRL (1)
certSerialNumber	not generated	
issuerAltName	supported	optional
cRLNumber	supported: included in all CRLs	optional
issuingDistributionPoint	supported	supported
deltaCRLIndicator	optional	optional

NOTES:

- For CRLs, "supported" means that the CA is capable of issuing CRLs that contain this extension.
- For Clients, "supported" means that the client is capable of processing this extension in CRLs.

(1) Clients shall be capable of finding the certificate used to sign a CRL, when the CA has multiple certificates, and the certificates are accessible in the appropriate directory, whether or not they use this extension to do so, and whether or not the CRL contains this extension.

Reason Code

The **reasonCode** is a noncritical CRL entry extension that identifies the reason for the certificate revocation. CAs shall be capable of generating this extension in CRL entries. Processing of the **reasonCode** extension by clients is optional, that is clients shall not validate a certificate if any certificate in the certification path is listed in a current CRL, regardless of the **reasonCode,** and need not provide operator information about the reason for failure. The following enumerated **reasonCode** values are defined:

- **unspecified**; this value shall not be used;
- **keyCompromise** indicates compromise or suspected compromise;
- **cACompromise** indicates that the certificate has been revoked because of a CA key compromise. It shall only be used to revoke CA certificates;
- **affiliationChanged** indicates that the certificate was revoked because of a change of affiliation of the certificate subject;
- **superseded** indicates that the certificate has been replaced by a more recent certificate;
- **cessationOfOperation** indicates that the certificate is no longer needed for the purpose for which it was issued, but there is no reason to suspect that the private key has been compromised.
- **certificateHold** indicates that the certificate shall not be used at this time. When clients process a certificate that is listed in a CRL with a **reasonCode** of **certificateHold**, they shall fail to validate the certification path.
- **removeFromCRL**, which is used only with delta-CRLs and indicates that an existing CRL entry should be removed.

Expiration Date

The **expirationDate** is a noncritical CRL entry extension that indicates the expiration of a hold entry in a CRL. This extension shall not be used in CRLs or by clients.

Instruction Code

The **instructionCode** is a noncritical CRL entry extension that provides a registered instruction identifier which indicates the action to be taken after encountering a certificate that has been placed on hold. This extension shall not be used in CRLs.

Invalidity Date

The **invalidityDate** is a noncritical CRL entry extension that provides the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid. This date may be earlier than the revocation date in the CRL entry. The revocation date in the CRL entry specifies the date that the CA revoked the certificate. Whenever this information is available, CAs are encouraged to share it with CRL users. CAs shall be capable of generating this extension in CRLs. This value is represented as **GeneralizedTime**.

Certificate Issuer

The **certificateIssuer** CRL entry extension is used with an indirect CRL (a CRL that has the **indirectCRL** indicator set in its **issuingDistributionPoint** extension). If this extension is not present in the first entry of an indirect CRL, the certificate issuer defaults to the CRL issuer. In subsequent entries in an indirect CRL, when the certificateIssuer extension is not present, the certificate issuer is the same as the issuer of the preceding CRL entry.

Summary of CRL Entry Extension Use

Table 3-5 summarizes the CRL entry extensions while Table 3-6 summarizes the use of CRL entry extensions for the MISPC.

Table 3-5 Summary of CRL Entry Extensions

Extension	Use	Critical
reasonCode	identifies the reason for the revocation of this certificate	No
instructionCode	used with **certificateHold reasonCode**; indicates action to be taken when encountering a held certificate	No
invalidityDate	date certificate became invalid	No
certificateIssuer	Issuer of revoked certificate in an indirect CRL	Yes

Table 3-6 Summary of CRL Entry Extensions Use for MISPC

Extension	CRL	Clients
reasonCode	supported; included for all entries	optional - may be used to provide information about validation failure
instructionCode	not used	optional
invalidityDate	supported	optional - may be used to provide information about validation failure
certificateIssuer	optional	optional - necessary to support processing of indirect CRLs

NOTES

For CRLs, “supported” means that CAs are capable of issuing CRLs that contain this CRL entry extension.
For clients, “supported” means that the client is capable of processing this entry extension in CRLs.

3.3 Certification Path Validation

The procedure specified in section 12.4.3 of the DAM [DAM], Certification path processing procedure, shall be adopted by clients.

3.4 Transaction Message Formats

This section presents a set of message formats to support the minimal set of PKI transactions. Systems that implement these transactions shall support these message formats, generating and recognizing them as appropriate. The message formats are specified in ASN.1; messages shall be encoded and transmitted using the Distinguished Encoding Rules (DER).

These message formats are used to implement transactions described in section 3.5.

3.4.1 Overall PKI Message Components

PKI Message

Each message has three components

```
PKIMessage ::= SEQUENCE {
     header               PKIHeader,
     body                 PKIBody,
     protection           [0]   PKIProtection OPTIONAL,
     extraCerts           [1]   SEQUENCE OF Certificate OPTIONAL
 }
```

The extraCerts field is not used within this specification.

PKI Message Header

All PKI messages require some header information for addressing and transaction identification. Some of this information will also be present in a transport specific envelope, however, if the PKI message is signed then this information is also protected (i.e., we make no assumption about secure transport).

The following data structure is used to contain this information:

```
PKIHeader ::= SEQUENCE {
    pvno                 INTEGER              { fpki-version1 (0) },
    sender               GeneralName,         -- identifies the sender
    recipient            GeneralName,         -- identifies the intended recipient
    messageTime          [0] GeneralizedTime           OPTIONAL,
    -- time of production of this message (used when sender)
    -- that the time will still be meaningful upon receipt)
    protectionAlg        [1] AlgorithmIdentifier                OPTIONAL,
    -- algorithm used for calculation of protection bits
    senderKID            [2] KeyIdentifier             OPTIONAL,
    recipKID             [3] KeyIdentifier             OPTIONAL,
    -- to identify specific keys used for protection
    transactionID        [4] OCTET STRING              OPTIONAL,
    -- identifies the transaction, i.e., this will be the same in corresponding
    -- request, response and confirmation messages
    senderNonce          [5] OCTET STRING              OPTIONAL,
    recipNonce           [6] OCTET STRING              OPTIONAL,
    -- nonces used to provide replay protection, senderNonce is inserted by the creator
    -- of this message; recipNonce is a nonce previously inserted in a related message by
    -- the intended recipient of this message
    freeText             [7] PKIFreeText                OPTIONAL
    -- this may be used to indicate context-specific instructions (this field is intended for
    -- human consumption)
  }
```

```
PKIFreeText ::= CHOICE {
   iA5String  [0] IA5String,
   bMPString  [1] BMPString)
}
```

The **transactionID** field within the message header allows the recipient of a response message to correlate this with the request. In the case of an ORA there may be many requests "outstanding" at a given moment. The value of this field should be unique from the sender's perspective in order to be useful.

The **messageTime** field indicates the time the message was generated. The value included in this field shall be expressed Greenwich Mean Time (Zulu) and shall include seconds (i.e., times are YYYYMMDDHHMMSSZ), even where the number of seconds is zero. The **messageTime** values shall not include fractional seconds.

The **sender** and **recipient** fields within the message header are defined as **GeneralName**. Systems are required to support X.500 distinguished names and RFC 822 (Internet electronic mail) names.

The freetext field is defined as PKIFreeText, which may be an IA5String (basically, ASCII) or BMPString. For this specification, PKIFreeText will always be an IA5String.

The **protectionAlg** is required for all signed messages. The **senderNonce**, **recipNonce, senderKID,** and **recipKID** fields are not required to implement this specification.

PKI Message Body

```
PKIBody ::= CHOICE {
    -- message-specific body elements
   cr        [2]    CertReqContent,
   cp        [3]    CertRepContent,
   p10cr     [4]    PKCS10CertReqContent,
   rr        [11]   RevReqContent,
   rp        [12]   RevRepContent,
   conf      [19]   PKIConfirmContent,
   }
```

Additional message-specific body elements are defined by [PKIX3]. The additional elements are not required to implement this specification, so they were omitted for clarity. The complete list of message-specific body elements appears in Appendix C.

Other sections of this document refer to **CertReq, CertRep, RevReq**, and **RevRep** messages. These terms refer to PKIMessages with body elements **cr, cp, rr**, and **rp**, respectively. A PKCS #10 request refers to a message with a **p10cr** body element. A confirmation message will have body element **conf**.

PKI Message Protection

All PKI messages will be protected for integrity using the following structure:

```
PKIProtection ::= BIT STRING
```

The input to the calculation of the **PKIProtection** is the DER encoding of the following data structure:

```
ProtectedPart ::= SEQUENCE {
        header          PKIHeader,
        body            PKIBody }
```

In most cases, the **PKIProtection** field will contain a digital signature and the **protectionAlg** field in the **PKIHeader** will contain an **AlgorithmIdentifier** specifying the digital signature algorithm (e.g., dsaWithSha-1) used to protect the message.

In some cases, such as key update, it may be necessary to attach multiple signatures. In this case, signed messages are nested - each signed message becomes a **PKIBody** element **nested**; the next signature is applied to this message. This process is repeated until all signatures have been applied.

Where symmetric techniques are needed for message authentication, the algorithm id shall be one of those identified in section 3.1.2.2 and the **protectionBits** value shall contain the message authentication code using the DER encoded header and body as input (and the shared secret as the DES key.) The **PKIHeader** will contain an **AlgorithmIdentifier** specifying a message authentication code algorithm (e.g., DES-MAC).

3.4.2 Common Data Structures

The following data types are common to several message formats.

Certificate Templates

In various PKI management messages, the originator may provide certain values to identify an existing certificate or request certain values be used in the generation of a certificate. The **CertTemplate** structure allows entities to indicate those values. **CertTemplate** includes all the same information as a certificate.

```
CertTemplate ::= SEQUENCE {
        version         [0] Version                 OPTIONAL,
                -- used to ask for a particular syntax version
        serial          [1] INTEGER                 OPTIONAL,
                -- used to ask for a particular serial number or to indicate request
                -- is on behalf of a previous certificate holder
        signingAlg      [2] AlgorithmIdentifier     OPTIONAL,
        subject         [3] Name                    OPTIONAL,
        validity        [4] OptionalValidity        OPTIONAL,   -- policy
        issuer          [5] Name                    OPTIONAL,
        publicKey       [6] SubjectPublicKeyInfo    OPTIONAL,  -- required
        issuerUID       [7] UniqueIdentifier        OPTIONAL,   -- not supported
        subjectUID      [8] UniqueIdentifier        OPTIONAL,   -- not supported
        extensions      [9] Extensions              OPTIONAL,
```

```
                    -- contains the extensions which the requester
                    -- would like in the cert.
}

OptionalValidity ::= SEQUENCE {
        notBefore       [0] UTCTime OPTIONAL,
        notAfter        [1] UTCTime OPTIONAL
}

CertTemplates ::= SEQUENCE OF CertTemplate
```

If it appears, the **validity** field contains the requested issuance date (in the **notBefore** field) and expiration date (**notAfter**) for the requested certificate. The **UTCTime** values in the **CertTemplate validity** field shall be interpreted as specified for the certificate **validity** field (see sec. 3.1.1).

Proving Possesion of a new Signature Key

Conforming CAs verify that the prospective subject of a certificate request holds the private key corresponding to the public key provided in a certificate request. This is performed with the following **POPOSigningKey** structure. This structure includes input data, an algorithm identifier, and a signature. The input data is constrained to match the data in the certificate request, and includes the public key itself.

```
POPOSigningKey ::= SEQUENCE {
   poposkInput          POPOSKInput,
   alg                  AlgorithmIdentifier,
   signature            BIT STRING
   -- the signature (using "alg") on the DER-encoded
   -- value of poposkInput
}

POPOSKInput ::= CHOICE {
   popoSigningKeyInput       [0] POPOSigningKeyInput,
   certificationRequestInfo    CertificationRequestInfo
   -- imported from [PKCS10] (note that if this choice is used,
   -- POPOSigningKey is simply a standard PKCS #10 request; this
   -- allows a bare PKCS #10 request to be augmented with other
   -- desired information in the FullCertTemplate before being
   -- sent to the CA/RA)
}

POPOSigningKeyInput ::= SEQUENCE {
   authInfo          CHOICE {
      sender              [0] GeneralName,
      -- from PKIHeader (used only if an authenticated identity
      -- has been established for the sender (e.g., a DN from a
      -- previously-issued and currently-valid certificate)
      publicKeyMAC   [1] BIT STRING
      -- used if no authenticated GeneralName currently exists for
      -- the sender; publicKeyMAC contains a password-based MAC
      -- (using the protectionAlg AlgId from PKIHeader) on the
      -- DER-encoded value of publicKey
   },
   publicKey         SubjectPublicKeyInfo    -- from CertTemplate
```

```
}
```

FullCertTemplates

The FullCertTemplate augments the CertTemplate structure with a certificate request id and four optional fields. The optional fields are not used within this specification.

The **FullCertTemplates** structure is a sequence of a **FullCertTemplate**. This structure permits "batch processing" of requests in a single transaction. Since this may also be performed through a series of transactions, this feature is not supported in this specification. **FullCertTemplates** may be considered a sequence of exactly one **FullCertTemplate** wherever it appears.

```
FullCertTemplates ::= SEQUENCE OF FullCertTemplate

FullCertTemplate ::= SEQUENCE {
   certReqId          INTEGER,
   -- to match this request with corresponding response
   -- (note:  must be unique over all FullCertReqs in this message)
   certTemplate       CertTemplate,
   popoSigningKey     [0] POPOSigningKey   OPTIONAL,
   archiveOptions     [1] PKIArchiveOptions OPTIONAL,     -- not used in this specification
   publicationInfo    [2] PKIPublicationInfo OPTIONAL,    -- not used in this specification
   oldCertId          [3] CertId           OPTIONAL
        -- id. of cert. which is being updated by this one
}
```

Status codes for PKI messages

All response messages will include some status information. The following values are defined:

```
PKIStatus ::= INTEGER {
        granted                 (0),
                -- request granted without change
        grantedWithMods         (1),
                -- request granted, with modifications; the  requester
                -- is responsible for ascertaining the differences
        rejection               (2),
                -- request rejected
        waiting                 (3),
                -- the request has been received but has not been processed,
                -- an additional response will follow after processing
        revocationWarning       (4),
                -- this message contains a warning that a revocation has
                -- been requested and is under consideration
        revocationNotification (5),
                --  notification that a revocation has occurred
        keyUpdateWarning (6)
                --
}
```

This specification does not use the status code **keyUpdateWarning**.

Failure Information

Responders use the following syntax to provide more information about failure cases.

```
PKIFailureInfo ::= BIT STRING {    -- since we can fail in more than
                        -- one way!
        badAlg              (0),    -- unrecognized  or unsupported algorithm identifier
        badMessageCheck     (1),    -- integrity check failed (e.g., signature did not verify)
        badRequest          (2),    -- transaction not permitted or supported
        badTime             (3),    -- messageTime field was not sufficiently close
                                    -- to the system time, as defined by local policy
        badCertId           (4),    -- no certificate could be identified matching the
                                    -- provided criteria
        badPoP              (5)     -- proof of possession field did not verify
        -- need more failure information
}

PKIStatusInfo ::= SEQUENCE {
                status          PKIStatus,
                statusString    PKIFreeText         OPTIONAL,
                failInfo        PKIFailureInfo      OPTIONAL
 }
```

Protocol Confirmation

Confirmation messages shall carry all the required information in the **PKIHeader**. As a result, this data structure has a NULL content.

```
PKIConfirmContent ::= NULL
```

Certificate Identification

In order to identify particular certificates the **CertId** structure is used.

```
CertId ::= SEQUENCE {
        issuer          GeneralName,
        serialNumber    INTEGER
}
```

Out-of-band Information

To convey a CA's public key out of band, OOBCert structure is used. OOBCert is simply the CA's certificate.

```
OOBCert ::= Certificate
```

3.4.3 Operation-Specific Data Structures

Registration/Certification Request

Registration/Certification request message (**cr**) contains a **CertReqContent** data structure which specifies values for one or more requested certificates.

```
CertReqContent ::= FullCertTemplates
```

The certificate request body shall include the prospective certificate holder's distinguished name and public key in the **subject** and **publicKey** fields.

Registration/Certification Response

A registration response message (**CertRep**) contains a **CertRepContent** structure which is an optional CA public key and a **response.** The response is a sequence of **CertResponse**; for the purposes of this specification, **response** is considered a sequence of exactly one **CertResponse**. The **CertResponse** includes a request id, status information and optionally a **CertifiedKeyPair**. The **CertifiedKeyPair** is a sequence of four optional fields: a certificate, an encrypted certificate, an encrypted private key, and publication information. In this specification, the certificate field will always appear in **CertifiedKeyPair** but the other fields are never present.

```
  CertRepContent ::= SEQUENCE {
        caPub           [1] Certificate    OPTIONAL,
        response        SEQUENCE OF CertResponse
 }

 CertResponse ::= SEQUENCE {
       certReqId                  INTEGER,       -- to match this response with corresponding
request
        certRepStatus             PKIStatusInfo,
        certifiedKeyPair          CertifiedKeyPair        OPTIONAL       -- present if status is granted
                                                                         -- or grantedWithMods
 }

CertifiedKeyPair ::= SEQUENCE {
        certificate         [0] Certificate            OPTIONAL,    -- required for this specification
        encryptedCert    [1] EncryptedValue    OPTIONAL,    -- not used in this specification
        privateKey         [2] EncPrivKey          OPTIONAL,    -- not used in this specification
        publicationInfo  [3] PKIPublicationInfo OPTIONAL    -- not used in this specification
 }
```

If **certRepStatus** contains a **failInfo** field, the **CertResponse** shall not include a **certifiedKeyPair** and the value in the **certRepStatus** field shall be **rejection** on the value of status. For the status value **waiting** none of the optional fields may be present. The status values **revocationWarning** and **revocationNotification** should not appear in this message.

The **caPub** field is not required, and may be ignored if present. This interoperability specification does not use the **encryptedCert**, **privateKey**, and **encryptedCert** fields in **CertifiedKeyPair**.

Revocation Request Content

When requesting revocation of a certificate the following data structure is used. The name of the requester is present in the **PKIHeader** structure.

```
RevReqContent ::= SEQUENCE  OF RevDetails

RevDetails ::= SEQUENCE  OF {
     certDetails                    CertTemplate,
         -- allows requester to specify as much as they can about
         -- the cert. for which revocation is requested
         -- (e.g. for case serialNumber not available)
     revocationReason            ReasonFlags,
         -- from the DAM, so that CA knows which Dist. point to use
     badSinceDate                 GeneralizedTime    OPTIONAL,
```

```
        -- indicates best knowledge of sender
    crlEntryDetails              Extensions}
        -- requested crlEntryExtensions
```

ReasonFlags are defined in Appendix B. but are reproduced here for clarity.

```
ReasonFlags ::= BIT STRING {
        unused                              (0),
        keyCompromise               (1),
        caCompromise                (2),
        affiliationChanged          (3),
        superseded                  (4),
        cessationOfOperation        (5),
        certificateHold             (6),
        removeFromCRL               (8) }
```

Revocation Response Content

The response to the above message. If produced this is sent to the requester of the revocation.

```
RevRepContent ::= SEQUENCE {
        status          PKIStatusInfo,
        revCerts        [0] SEQUENCE OF CertId OPTIONAL,
                 -- identifies the cert for which revocation
                -- was requested
        crls            [1] SEQUENCE OF CertificateList    OPTIONAL}
                -- the resulting CRL
```

For the purposes of this specification, revCerts shall be a SEQUNCE of one CertId, and the crls field does not appear.

PKCS #10 Certification Request

This alternative certification request syntax is defined in [PKCS#10]. It is reproduced here for clarity.

```
PKCS10CertReqContent ::= SEQUENCE {
        certificationRequestInfo        CertificationRequestInfo
        signatureAlgorithm              SignatureAlgorithmIdentifier,
        signature                       Signature
 }

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

Signature ::= BIT STRING

CertificationRequestInfo::= SEQUENCE {
        version                 Version,
        subject                 Name,
        subjectPublicKeyInfo    SubjectPublicKeyInfo,
        attributes              [0] IMPLICIT Attributes
  }

Version ::= INTEGER
```

Attributes ::= SET OF Attribute

Attributes are specified in [PKCS#9]. Support for attributes is optional for conforming implementations. If present, they may be ignored.

3.5 PKI Transactions

This section describes PKI specific functions to request, renew, or revoke certificates. This section also provides a brief description of transactions for accessing the directory service.

Compliant CAs shall implement all of the transactions identified in this section. Compliant ORAs shall implement the ORA-Generated Registration (sec. 3.5.1) and Request Revocation (sec. 3.5.5) transactions. Compliant certificate holders shall implement the Request Revocation (sec. 3.5.5) and ORA-Generated Registration (sec. 3.5.1) transactions. Self Registration (secs. 3.5.3 and 3.5.4) and Certificate Renewal (sec. 3.5.2) transactions are optional for certificate holders.

3.5.1 ORA-Generated Registration Requests

An ORA may request that a CA issue a certificate for an end entity. This transaction is performed in three steps. In the first step, the end entity provides a public key to the ORA in a signed message in an out-of-band transaction (e.g., by physically presenting a diskette). In the second step, the ORA requests a certificate from the CA in a signed message. The CA replies to the ORA with a signed message containing either a certificate or an error message. The ORA provides the end entity with the CA's public key out-of-band. The end entity may receive the certificate from the ORA out-of-band, or from the CA electronically.

<u>Certificate Request from an End Entity to the ORA</u>

The end entity creates a **PKIMessage** with **PKIBody** element **cr**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the end entity, or null;
- **recipient** is the distinguished name of the ORA, or null; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The message body is **CertReqContent**, which is a sequence of one or more **FullCertTemplate**. For these specifications, **CertReqContent** is a sequence of one **FullCertTemplate**. The **FullCertTemplate** will include the following information:

- **certReqID** is any integer;
- **certTemplate** is a **CertTemplate** including, at a minimum, the **publicKey** field which provides the public key for the new certificate; and
- **popoSigningKey** provides proof of possession of the private key for the new certificate.

Optionally, **oldCertID** identifies a current or expired certificate for this subject issued by the same CA. If the **oldCertID** field is omitted, this indicates that the end entity has not previously held a certificate issued by this CA. If it appears, the **oldCertID** identifies a certificate previously issued to this entity by **recipient**. In this case, that certificate's subject distinguished name should be used as the subject of the new certificate.

The following information may be included in the **CertTemplate**:

- **signingAlg** specifies the preferred signature algorithm;
- **subject** specifies the distinguished name for the prospective certificate holder;

The **popoSigningKey** field shall be generated using the private key corresponding to the public key in the **publicKey** field.

If the end entity is a current certificate holder, the **PKIProtection** field contains the end entity's signature, calculated on the DER encoded sequence of the header and body with private key material corresponding to the current certificate. If the end entity is not a current certificate holder, the **PKIProtection** field shall be an empty string..

<u>Certificate Request from ORA to CA</u>

The ORA creates a **PKIMessage** with **PKIBody** element **cr**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **transactionID** is an integer unique to this transaction for this ORA;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the ORA;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The message body is **CertReqContent**, which is a sequence of one or more **FullCertTemplate**. For these specifications, **CertReqContent** is a sequence of one **FullCertTemplate**. The **FullCertTemplate** will include the following information:

- **certReqID** is an integer;
- **certTemplate** is a **CertTemplate** (a **SEQUENCE** whose contents are described below); and
- **popoSigningKey** provides proof of possession of the private key for the new certificate.

Optionally, **oldCertID** identifies a current or expired certificate for this subject issued by the same CA. If the **oldCertID** field is omitted, this indicates that the end entity has not previously held a certificate issued by this CA. If it appears, the **oldCertID** identifies a certificate previously issued to this entity by **recipient**. In this case, that certificate's subject distinguished name should be used as the subject of the new certificate.

The **CertTemplate** will include the following information:

- **version** is v3 (2);
- **publicKey** provides the public key for the new certificate; and
- **extensions** specifies, at a minimum, the certificate policy OID to be associated with the certificate.

The following information may be included in the **CertTemplate**:

- **signingAlg** specifies the preferred signature algorithm;
- **subject** specifies the distinguished name for the prospective certificate holder;

If **signingAlg** does not appear, the CA should sign with the algorithm corresponding to the entity's public key.

The request shall not include the following information:

- **issuerUID**; and
- **subjectUID**.

The **popoSigningKey** field shall be the same as provided in the request delivered to the ORA.

The **PKIProtection** field contains the ORA's signature, calculated on the DER encoded sequence of the header and body.

<u>Certificate Response from CA to ORA</u>

The CA will return a **PKIMessage** with **PKIBody** element **cp** to the ORA.

The **PKIHeader** includes the following information:

- **pvno** is zero;
- **transactionID** is the same as the transactionID field in the cr message;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the CA;
- **recipient** is the distinguished name of the ORA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

If a **senderNonce** was supplied in the certificate request message, the header of the response shall include it as **recipNonce**.

The **PKIBody** element **cp** is of type **CertRepContent**. If the CA issued a certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **certificate** will contain the X.509 version 3 certificate.

The certificate must meet the following properties:

- version number shall be v3 (2);
- The **publicKey** field shall be the same as in the certificate request;
- the subject distinguished name shall be the same as in the certificate request;
- the issuer name shall be the CA's distinguished name;
- if **notBefore** was present in the certificate request, the certificate shall be valid from the issuance date or the **notBefore** date, whichever is later; and
- if **notAfter** was present in the certificate request, the certificate shall expire on or before that date.

The certificate shall contain the following extensions:

- a **subjectKeyIdentifier** field;
- at least one certificate policy OID in the **certificatePolicies** field; and
- an authority key identifier including a **KeyIdentifier** field.

If a specific key identifier was specified in the certificate request message, the certificate shall contain that key identifier as the **subjectKeyIdentifier** field. If no key identifier was supplied, the CA shall use the 160-bit SHA-1 hash of the subject public key as the **keyidentifier** in the **subjectKeyIdentifier** field. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the certificate.

The certificate shall include URLs in the **issuerAltName** extension and **distributionPoint** field of the **CRLDistributionPoints** extension if the issuer's certificates or CRLs are not available from a well known X.500 directory.

The **failInfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failInfo** will contain the appropriate failure codes:
  - **badAlg** indicates that the CA cannot validate the signature because the algorithm identifier is unrecognized or unsupported;
  - **badMessageCheck** indicates that the signature in the **PKIProtection** field was checked but did not match;
  - **badPoP** indicates that the signature in the **popoSigningKey** field was checked but did not match;
  - **badRequest** indicates that the responder does not permit or support the transaction;
  - **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time;[19] and
  - **badCertId** indicates that no certificate could be identified matching the nonzero **serial** field, or that the certificate was not issued by this CA.

The **certificate** field may not be present if **status** is **rejected**.

The **PKIProtection** field contains the CA's signature, calculated on the DER encoded sequence of the header and body.

3.5.2 Certificate Renewal Request

An entity that is a current certificate holder may request issuance of a new certificate directly from the CA that issued the current certificate. The requesting entity creates a PKI **cr** (certificate request) message requesting a certificate and includes proof of possession of the private key corresponding to the public key in the certificate request. The entity then signs the message with the private key corresponding to the entity's unexpired, unrevoked certificate.

---

[19] This error code assumes a locally defined window of time for responding to a PKI message. The MISPC does not require such a policy, but defines this error code to support such policies.

If the CA's Certificate Practice Statement permits certificate renewal,[20] it will return a **cp** (certificate response) message to the certificate holder. This message will contain the certificate or a reason code for the transaction failure.

Certificate Renewal Request from Certificate Holder to CA

The certificate holder creates a key update request: a **PKIMessage** with **PKIBody** element **cr**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the certificate holder;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The message body is **CertReqContent**, which is a sequence of one or more **FullCertTemplate**. For these specifications, **CertReqContent** is a sequence of one **FullCertTemplate**. The **FullCertTemplate** will include the following information:

- **certReqID** is an integer;
- **certTemplate** is a **CertTemplate** (a **SEQUENCE** whose contents are described below);
- **popoSigningKey** provides proof of possession of the private key for the new certificate; and
- **oldCertID** identifies a current certificate for this subject issued by the same CA.

The **CertTemplate** will include the following information:

- **version** is v3 (2); and
- **publicKey** provides the public key for the new certificate.

The following information may be included in the **CertTemplate**:

- **signingAlg** specifies the preferred signature algorithm;
- **subject** specifies the distinguished name for the prospective certificate holder;

If **signingAlg** does not appear, the CA should sign with the algorithm corresponding to the entity's public key.

The request shall not include the following information:

- **issuerUID**; and
- **subjectUID**.

The **PKIProtection** field contains a signature generated using the private key associated with the current unexpired, unrevoked certificate and calculated upon the DER encoded sequence of the header and body.

---

[20] Conforming CA *implementations* shall support certificate renewal. However, a particular CA may choose not to support this transaction as a matter of policy.

Certificate Renewal Response from CA to Certificate Holder

The CA will return a key update response (a **PKIMessage** with **PKIBody** element **cp**) message to the certificate holder.

The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the CA;
- **recipient** is the distinguished name of the certificate holder and the **sender** of the **cr** message; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

If a **transactionID** was supplied in **cr** message, the header of the response will include the same **transactionID**. If a **senderNonce** was supplied in the **senderNonce** message, the header of the response shall include it as **recipNonce**.

The **PKIBody** is the element **cp** and is of type **CertRepContent**. If the CA issued a certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **certificate** will contain the new X.509 version 3 certificate.

The certificate shall contain the following extensions:

- a **subjectKeyIdentifier** field;
- at least one certificate policy OID in the **certificatePolicies** field; and
- an authority key identifier including a **KeyIdentifier** field.

The **certificatePolicies** extension shall be identical to that found in the certificate identified in the **cr** message's **oldCertID** field. If a specific key identifier was specified in the **cr** message, the certificate shall contain that key identifier as the **subjectKeyIdentifier** field. If no key identifier was supplied, the CA shall use the 160-bit SHA-1 hash of the subject public key as the **keyidentifier** in the **subjectKeyIdentifier** field. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the certificate.

If the **cr** message included extensions other than the **subjectKeyIdentifier**, the CA may modify or ignore the requested extensions.

The certificate shall include URLs in the **issuerAltName** extension and **distributionPoint** field of the **CRLDistributionPoints** extension if the issuer's certificates or CRLs are not available from a well known X.500 directory.

The **failInfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failInfo** will contain the appropriate failure codes:

- **badAlg** indicates that the CA cannot validate the signature because the algorithm identifier is unrecognized or unsupported;
- **badPoP** indicates the signature in the **popoSigningKey** field was checked but did not match;
- **badMessageCheck** indicates that the signature in the **PKIProtection** field was checked but did not match;
- **badRequest** indicates that the responder does not permit or support the transaction;
- **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time; and
- **badCertId** indicates that no certificate could be identified matching the nonzero **serial** field.

The **certificate** field may not be present if **status** is **rejected**.

The **PKIProtection** field contains the CA's signature, calculated on the DER encoded sequence of the header and body.

3.5.3 Self-Registration Request

An entity that is not a current certificate holder may request issuance of a new certificate directly from the CA that issued the current certificate. The requesting entity creates a **PKIMessage cr** requesting a certificate and include proof of possession of the private key corresponding to the public key in the certificate request. The entity protects the message with a DES-MAC using a secret key provided by the ORA.

If the CA supports certificate renewal, it will return a **cp** message to the certificate holder. This message will contain the certificate or a reason code for the transaction failure.

<u>ORA-Entity Out-of-Band Transaction</u>

The self-registration request for a certificate begins with exchange of a secret known to the ORA to the entity requesting a certificate. This information will allow the entity to authenticate themselves to the CA through generation of a message authentication code from the shared secret.

The precise content and format of this out-of-band transaction are not specified. However, it should be noted that both the secret key and the public key material for the trusted CA must be conveyed to the entity in a trusted fashion. So, this transaction should include authentication information for the CA of whom the certificate will be requested and the public key material for the trusted CA.

<u>Self-Registration Request from Certificate Holder to CA</u>

The requester creates a **PKIMessage** with a **PKIBody** element **cr**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the (proposed) distinguished name of the requester or an electronic mail address;

- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The message body is **CertReqContent**, which is a sequence of one or more **FullCertTemplate**. For these specifications, **CertReqContent** is a sequence of one **FullCertTemplate**. The **FullCertTemplate** will include the following information:

- **certReqID** is an integer;
- **certTemplate** is a **CertTemplate** (a **SEQUENCE** whose contents are described below); and
- **popoSigningKey** provides proof of possession of the private key for the new certificate.

Optionally, **oldCertID** identifies a current or expired certificate for this subject issued by the same CA. If the **oldCertID** field is omitted, this indicates that the end entity has not previously held a certificate issued by this CA. If it appears, the **oldCertID** identifies a certificate previously issued to this entity by **recipient**.[21] In this case, that certificate's subject distinguished name should be used as the subject of the new certificate.

The **CertTemplate** will include the following information:

- **version** is v3 (2); and
- **publicKey** provides the public key for the new certificate.

The following information may be included in the **CertTemplate**:

- **signingAlg** specifies the preferred signature algorithm;
- **subject** is present if and only if serial equals zero, and specifies the distinguished name for the prospective certificate holder; and
- **extensions** requests a particular certificate policy OID be specified in the certificate.

The request shall not include the following information:

- **issuerUID**; and
- **subjectUID**.

The **PKIProtection** field contains a value that is generated by the requester using the secret value obtained from the ORA. The entity generates a 32 bit DES-MAC using the secret key provided by the ORA. The **protectionAlg** field shall be set to DES-MAC, and the value of **PKIprotection** shall be the 32 bit message authentication code. The input to the calculation of the **PKIprotection** is the DER encoding of the following data structure:

```
ProtectedPart ::= SEQUENCE {
      PKIHeader,
      PKIBody}
```

<u>Self-Registration Request Response from CA to Certificate Requester</u>

The CA will return a **PKIMessage** with a **PKIBody** element **cp** to the certificate holder.

[21] If serial is nonzero, the entity is renewing their certificate but was not permitted to request the new certificate directly. This may be because of CA policy or because the entity's certificate was expired or revoked.

The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the CA;
- **recipient** is the value of **sender** in the certificate request header; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

If a **transactionID** was supplied in **cr** message, the header of the response will include the same **transactionID**. If a **senderNonce** was supplied in the **senderNonce** message, the header of the response shall include it as **recipNonce**.

The **PKIBody** is a **cp** element and is of type **CertRepContent**. If the CA issued a certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **certificate** will contain the new X.509 version 3 certificate;

The **failInfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failInfo** will contain the appropriate failure codes:
  - **badAlg** indicates that the CA cannot validate the signature because the algorithm identifier is unrecognized or unsupported;
  - **badPoP** indicates the signature in the **popoSigningKey** field was checked but did not match;
  - **badMessageCheck** indicates the MAC in the **PKIProtection** field was rejected;
  - **badRequest** indicates that the responder does not permit or support the transaction;
  - **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time; and
  - **badCertId** indicates that no certificate could be identified matching the nonzero **serial** field.

The **certificate** field may not be present if **status** is **rejected**. If present, the certificate shall conform to the profile presented in section 3.1.1.

The certificate shall contain the following extensions:

- a **subjectKeyIdentifier** field;
- at least one certificate policy OID in the **certificatePolicies** field; and
- an authority key identifier including a **KeyIdentifier** field.

If a specific key identifier was specified in the **cr** message, the certificate shall contain that key identifier as the **subjectKeyIdentifier** field. If no key identifier was supplied, the CA shall use the 160-bit SHA-1 hash of the subject public key as the **keyidentifier** in the **subjectKeyIdentifier** field. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the certificate.

If the **cr** message included extensions other than the subjectKeyIdentifier, the CA may modify or ignore the requested extensions.

The certificate shall include URLs in the **issuerAltName** extension and **distributionPoint** field of the **CRLDistributionPoints** extension if the issuer's certificates or CRLs are not available from a well known X.500 directory.

If a specific key identifier was specified in the cr message, the certificate shall contain that key identifier. If no key identifier was supplied the CA shall use the 160-bit SHA-1 hash of the subject public key shall be used as the **keyidentifier** in the **subjectKeyIdentifier**. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the certificate.

The **PKIProtection** field contains the CA's signature, calculated on the DER encoded sequence of the header and body.

3.5.4 PKCS #10 Self-Registration Request

An entity that is not a current certificate holder may request issuance of a certificate directly from the CA using the certificate request syntax defined in PKCS #10. The requesting entity creates a **PKIMessage** of type **PKCSReq** requesting a certificate and includes proof of possession of the private key corresponding to the public key in the body of the certificate request, and protects the **PKIMessage** using a secret key provided by the ORA in an out-of-band transaction.

The CA will return a certificate request response message to the certificate requester. This message will contain the certificate or a reason code for the transaction failure.

The out-of-band transaction with the ORA and the CA response are identical to the corresponding steps in the Self-Registration Request defined in section 3.5.3.

<u>Self registration Request from Certificate Holder to CA</u>

The requester creates a **PKIMessage** with a **PKIBody** element **p10cr**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the (proposed) distinguished name of the requester or an electronic mail address;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The **PKIBody** is a **PKIBody** element **p10cr** which is of type **PKCS10CertReqContent**. This type is a sequence of a **certificationRequestInfo**, a **signatureAlgorithm** and a **signature**. The **certificationRequestInfo** will include the following information:

- **version** is v3 (2);
- **subject** is present if and only if serial equals zero, and specifies the distinguished name for the prospective certificate holder; and

- **subjectPublicKeyInfo** provides the public key and corresponding algorithm identifier for the new certificate.

The **signatureAlgorithm** field contains the algorithm identifier associated with the private key used to generate the **signature** field; the signature is generated using the DER-encoded **certificationRequestInfo** as input.

The **PKIProtection** field contains a value that is generated by the requester using the secret value obtained from the ORA. The entity generates a 32 bit DES-MAC using the secret key provided by the ORA. The **protectionAlg** field shall be set to DES-MAC, and the value of **PKIProtection** shall be the 32 bit message authentication code. The input to the calculation of the **PKIProtection** is the DER encoding of the following data structure:

```
ProtectedPart ::= SEQUENCE {
        header          PKIHeader,
        body            PKIBody}
```

PKCS Certificate Request Response from CA to Certificate Requester

The CA will return a **PKIMessage** with a **PKIBody** element **cp** to the certificate holder.

The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the CA;
- **recipient** is the value of **sender** in the certificate request header; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

If a **transactionID** was supplied in **PKCSReq** message, the header of the response will include the same **transactionID**.

The **PKIBody** element is a **cp**, which is of type **CertRepContent**. If the CA issued a certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **certificate** will contain the new X.509 version 3 certificate.

If a specific key identifier was specified in the **cr** message, the certificate shall contain that key identifier. If no key identifier was supplied the CA shall use the 160-bit SHA-1 hash of the subject public key shall be used as the **keyidentifier** in the **subjectKeyIdentifier**. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the certificate.

The **failInfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failInfo** will contain the appropriate failure codes:

- **badAlg** indicates that the CA cannot validate the signature because the algorithm identifier is unrecognized or unsupported;
- **badPoP** indicates the signature in the **pk10cr's signature** field was checked but did not match;
- **badMessageCheck** indicates the MAC in the **PKIMessage**'s **PKIProtection** field was rejected;
- **badRequest** indicates that the responder does not permit or support the transaction; and
- **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time.

The **certificate** field shall not be present if **status** is **rejected**.

The certificate shall contain the following extensions:

- a **subjectKeyIdentifier** field;
- at least one certificate policy OID in the **certificatePolicies** field; and
- an authority key identifier including a **KeyIdentifier** field.

If a specific key identifier was specified in the **cr** message, the certificate shall contain that key identifier as the **subjectKeyIdentifier** field. If no key identifier was supplied, the CA shall use the 160-bit SHA-1 hash of the subject public key as the **keyidentifier** in the **subjectKeyIdentifier** field. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the certificate.

If the **cr** message included extensions other than the **subjectKeyIdentifier**, the CA may modify or ignore the requested extensions.

The certificate shall include URLs in the **issuerAltName** extension and **distributionPoint** field of the **CRLDistributionPoints** extension if the issuer's certificates or CRLs are not available from a well known X.500 directory.

The **PKIProtection** field contains the CA's signature, calculated on the DER encoded sequence of the header and body.

3.5.5 Request Revocation

Certificate holders may request revocation of their own certificates. To perform this function the certificate holder generates a **RevReq** message, signs it with the private key corresponding to the certificate to be revoked, and sends it to the CA. The **RevReq** message shall identify the certificate(s) to be revoked and the reason for the revocation. The CA responds with a **RevRep** message.

ORAs may request revocation of a certificate issued to an entity on behalf of the certificate holder or the certificate holder's organization. To perform this function, the ORA generates a **RevReq** message, signs it with the ORA's private key, and sends it to the CA. The ORA shall generate a pseudo-random number and shall place it in the **transactionID** field. The **RevReq** message shall include, at a minimum, the certificate serial number in the serial field of **certDetails** and a revocation reason code in the **revocationReason** field.

The CA will respond to the revocation requester with an **rp** (**RevRep)** message. If the **rr (RevReq)** message included a **transactionID**, the CA shall include its contents as the **transactionID** in the **rp** message. The **rp** message shall contain, at a minimum, the status of the request in the status field and identify the certificate for which revocation is requested in the **revDetails** field.

Revocation Request from ORA or Certificate Holder to CA

The ORA or the certificate holder creates a **PKIMessage** with a **PKIBody** element **rr**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **transactionID** is an integer unique to this transaction for this ORA or any integer for the end entity;
- **messageTime** is the current time with a granularity of seconds;
- **sender** is the distinguished name of the ORA or the certificate holder;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The **PKIBody** is **RevReqContent**, which is a sequence of **RevDetails**. **RevDetails** is a sequence of **CertDetails**, reason flags, and date and time of compromise or loss. **CertDetails** is defined as a **CertTemplate**. For this interoperability specification, **RevReqContent** is a sequence of one **RevDetails. CertDetails,** at a minimum, includes the following information:

- **serial**, which contains the serial number of the certificate; and
- **issuer**, which contains the distinguished name of the certificate issuer.

or

- **subject**, which contains the distinguished name of the certificate holder; and
- **issuer**, which contains the distinguished name of the certificate issuer.

CertDetails may also include a **subjectKeyIdentifier** in the extensions field.

The **RevDetails** shall also include a reason code, and may include **badSinceDate** to specify the time after which the certificate should not be trusted. The reason code may not be **removeFromCRL**.

The **PKIProtection** field contains the requester's signature, calculated on the DER encoded sequence of the header and body.

Revocation Response from CA to Requester

The CA will return a **PKIMessage** with a **PKIBody** element **rr** to the requester. [22]

The **PKIHeader** includes the following information:

- **pvno** is zero;
- **transactionID** is the same as the transactionID field in the CertReq message;
- **messageTime** is the current time with a granularity of seconds;

---

[22] If the requester is an ORA, the CA may optionally send the RevRep message to the certificate holder as well.

- **sender** is the distinguished name of the CA;
- **recipient** is the distinguished name of the ORA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

If a **senderNonce** was supplied in the **senderNonce** message, the header of the response shall include it as **recipNonce**.

The **PKIBody** is **RpContent**. If the CA revoked the certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **revDetails** will contain the **CertId**(s) of the revoked certificate(s);

The **failinfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failinfo** will contain the appropriate failure codes:
  - **badAlg** indicates that the CA cannot validate the signature because the algorithm identifier is unrecognized or unsupported;
  - **badMessageCheck** indicates that the signature in the **PKIProtection** fields was checked but did not match;
  - **badRequest** indicates that the responder does not permit or support the transaction;
  - **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time; or
  - **badCertId** indicates that the information in latestCerts did not identify an unexpired, unrevoked certificate.

If the certificate in question can be determined, **revDetails** will contain the **CertId** of the certificate whose revocation was rejected.

The **PKIProtection** field shall contain the CA's signature, calculated on the DER encoded sequence of the header and body.

If the CA generates CRLs, and the revocation request was accepted, the CRL entry shall have the following values:

- the serial number of the revoked certificate in the **userCertificate** field;
- the **revocationDate** shall be the day and time the revocation request was received;
- the **crlEntryExtensions** shall be present and include:
  - the **reasonCode** shall be the **reasonCode** found in the **RevDetails** field;
  - optionally, the **invalidityDate** extension may be the **badSinceDate** found in the **RevDetails** field, if provided;

3.5.6 Request Certificate from a Repository

Entities may request certificates from a repository using LDAP [RFC1777]. When using LDAP, the entity may request certificates from a repository service using the certificate pair match rule, as defined in [DAM] or as specified in a given LDAP URL [RFC1959] (e.g., the **issuerAltName** field.)

3.5.7 Request CRL from a Repository

Entities may request CRLs from a repository using LDAP, the certificate list match rule, and the algorithm identifier match rule, as defined in [DAM]. Entities may request CRLs from a repository using LDAP [RFC 1777]. When using LDAP, the entity may request CRLs from a repository service using the certificate pair match rule, as defined in [DAM] or as specified in a given LDAP URL [RFC1959] (e.g., the **distributionPoint** field in the **cRLDistributionPoints** extension.)

4. References

[CONOPS] *Public Key Infrastructure Technical Specification: Part C - Concept of Operations*, William E. Burr. Available from http://csrc.nist.gov/pki

[COR95] ISO/IEC JTC 1/SC 21, *Technical Corrigendum 2 to ISO/IEC 9594-8 : 1990 & 1993 (1995:E)*. July 1995.

[DAM] ISO/IEC JTC 1/SC 21, Draft Amendments DAM 4 to ISO/IEC 9594-2, DAM 2 to ISO/IEC 9594-6, DAM 1 to ISO/IEC 9594-7, and DAM 1 to ISO/IEC 9594-8 on Certificate Extensions, June 30, 1996.

[FIPS113] FIPS PUB 113, *Computer Data Authentication*, NIST, May 1985.

[FIPS180] FIPS PUB 180-1, *Secure Hash Standard*, NIST, April 1995.

[FIPS186] FIPS PUB 186, *Digital Signature Standard*, NIST, May 1994.

[FIPS46] FIPS PUB 46-2, *Data Encryption Standard*, December 1993.

[ISO94-8] ISO/IEC 9594-8 (1994), *Open Systems Interconnection - The Directory: Authentication Framework*. 1994. The 1994 edition of this document has been amended by the Draft Amendments [DAM] and a *Technical Corrigendum* [COR95].

[ISO25-1] ISO/IEC 8825-1 (1994), *Information Technology - ASN.1 Encoding Rules - Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. 1994.

[PKCS#1] PKCS #1: RSA Encryption Standard, Version 1.4, RSA Data Security, Inc., 3 June 1991. available at: http://www.rsa.com/pub/pkcs/

[PKCS#9] PKCS #9: Selected Attribute Types, Version 1.1, RSA Data Security, Inc., 1 November, 1993. available at: http://www.rsa.com/pub/pkcs/

[PKCS#10] PKCS #10: Certification Request Syntax Standard, Version 1.0, RSA Data Security, Inc., 1 November, 1993. available at: http://www.rsa.com/pub/pkcs/

[PKIX1] Internet Draft, *Internet Public Key Infrastructure Part I: X.509 Certificate and CRL Profile*, R Housley, W. Ford and D. Solo, July 1997. working draft “in progress” available at: ftp://ds.internic.net/internet-drafts/draft-ietf-pkix-ipki-part1-04.txt

[PKIX3] Internet Draft, *Internet Public Key Infrastructure Part III: Certificate Management Protocols*, C. Adams and S. Farrell, June 1997. working draft “in progress” available at: ftp://ds.internic.net/internet-drafts/draft-ietf-pkix-ipki3cmp-02.txt

[RFC822] RFC 822, *Standard for the Format of ARPA Internet Text Messages*, David H. Crocker, August 13, 1982.

[RFC1777] RFC 1777, *Lightweight Directory Access Protocol,* Ed Yeoung, Howes, and Killie. March 1995.

[RFC1959] RFC 1959, *An LDAP URL Format*, T Howes, and M.Smith. June 1996.

[STAB95] OIW*, Stable Implementation Agreements for Open Systems Interconnection Protocols: Part 12 - OS Security*. June 1995.

[X9.55] Draft American National Standard X9.55-1995, *Public Key Cryptography for the Financial Services Industry: Extensions to Public Key Certificates and Certificate Revocation Lists*, Nov. 11, 1995

[X9.57] Working Draft American National Standard X9.57-199x, *Public Key Cryptography for the Financial Services Industry: Certificate Management*, June 21, 1996

[X9.62] Working Draft American National Standard X9.62-199x, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm*, June 21, 1996

Appendix A - X.509 v3 Certificate ASN.1

```
AuthenticationFramework {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7) 2}
DEFINITIONS ::=
BEGIN

-- EXPORTS All --
-- The types and values defined in this module are exported for use in the other ASN.1
--  modules contained within the Directory Specifications, and for the use of other applications
--  which will use them to access Directory services.  Other applications may use them for
 -- their own purposes, but this will not constrain extensions and modifications needed to
 -- maintain or improve the Directory service.

IMPORTS
        id-at, informationFramework, upperBounds selectedAttributeTypes, basicAccessControl
                FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0) 2}
        Name, ATTRIBUTE
                FROM InformationFramework informationFramework
        ub-user-password
                FROM UpperBounds upperBounds
        AuthenticationLevel
                FROM BasicAccessControl basicAccessControl
        UniqueIdentifier
                FROM SelectedAttributeTypes selectedAttributeTypes   ;

-- types --


Certificate             ::=     SIGNED {SEQUENCE{
        version                         [0]     Version DEFAULT v1,
        serialNumber                    CertificateSerialNumber,
        signature                       AlgorithmIdentifier,
        issuer                          Name,
        validity                        Validity,
        subject                         Name,
        subjectPublicKeyInfo                    SubjectPublicKeyInfo}
        issuerUniqueIdentifier          [1]     IMPLICIT UniqueIdentifier OPTIONAL,
                                                ---if present, version must be  v1 or v2--
        subjectUniqueIdentifier         [2]     IMPLICIT UniqueIdentifier OPTIONAL,
                                                ---if present, version must be v1 or v2--
        extensions                      [3]     Extensions Optional
                                                --if present, version must be v3--}  }


 Version                        ::=     INTEGER {v1(0), v2(1), v3(2) }

CertificateSerialNumber         ::=     INTEGER
Algorithmidentifier             ::=     SEQUENCE{
        algorithm               ALGORITHM.&id({SupportedAlgorithms}),
        parameters              ALGORITHM.&Type  ({SupportedAlgorithms}{ @algorithm}) OPTIONAL }

--      Definition of the following information object is deferred, perhaps to standardized
```

```
--      profiles of to protocol implementation conformance statements. This set is required to
--      specify a table constraint on the Parameters component of Algorithmidentifier.
--      SupportedAlgorithms  ALGORITHM  ::=        { ...|... }

 Validity                       ::=     SEQUENCE{
        notBefore       ChoiceOfTime,
        notAfter        ChoiceOfTime }

ChoiceOfTime ::= CHOICE {
        utcTime                 UTCTime,
        generalTime             GeneralizedTime }

 SubjectPublicKeyInfo           ::=     SEQUENCE{
        algorithm               AlgorithmIdentifier,
        subjectPublicKey        BIT STRING}

Extensions              ::=     SEQUENCE OF Extension

Extension               ::=     SEQUENCE {
        extnId          EXTENSION.&id ({ExtensionSet}),
        critical        BOOLEAN DEFAULT FALSE,
        extnValue               OCTET STRING
                                -- contains a DER encoding of a value of type &ExtnType for the
                                -- extension object identified by extnId --

-- Definition of the following information object set is deferred, perhaps to
-- standardized profiles or to protocol implementation conformance statements.
-- The set is required to specify a table constraint on the critical component
-- of Extension.
--      ExtensionSet   EXTENSION    ::=       { ... | ... }

EXTENSION    ::=        CLASS
{
        &id             OBJECT IDENTIFIER UNIQUE,
        &ExtnType
}
WITH SYNTAX
{
        SYNTAX                  &ExtnType
        IDENTIFIED BY           &id
}

Certificates            ::=             SEQUENCE {
        certificate                     Certificate,
        certificationPath               ForwardCertificationPath OPTIONAL}

ForwardCertificationPath        ::=     SEQUENCE OF CrossCertificates

CertificationPath               ::=     SEQUENCE {
        userCertificate                 Certificate,
        theCACertificates               SEQUENCE OF CertificatePair OPTIONAL}

CrossCertificates               ::=     SET OF Certificate
```

```
CertificateList ::=                 SIGNED { SEQUENCE {
        version                             Version  OPTIONAL, -- if present, must be v2
        signature                   AlgorithmIdentifier,
        issuer                      Name,
        thisUpdate                  ChoiceOfTime,
        nextUpdate                  ChoiceOfTime OPTIONAL,
        revokedCertificates         SEQUENCE OF SEQUENCE {
                userCertificate             CertificateSerialNumber,
                revocationDate              ChoiceOfTime,
                crlEntryExtensions          Extensions OPTIONAL } OPTIONAL,
        crlExtensions       [0]     Extensions OPTIONAL }}

CertificatePair     ::=     SEQUENCE {
        forward     [0]     Certificate OPTIONAL,
        reverse     [1]     Certificate OPTIONAL
                            -- at least one of the pair shall be present -- }

-- attribute types--

userPassword        ATTRIBUTE   ::=     {
        WITH SYNTAX                 OCTET STRING (SIZE  (0..ub-user-password))
        EQUALITY MATCHING RULE      octetStringMatch
        ID                          id-at-userPassword }

userCertificate     ATTRIBUTE   ::=     {
        WITH SYNTAX         Certificate
        ID                  id-at-userCertificate }

cACertificate       ATTRIBUTE   ::=     {
        WITH SYNTAX         Certificate
        ID                  id-at-cACertificate }

authorityRevocationList     ATTRIBUTE   ::=     {
        WITH SYNTAX         CertificateList
        ID                  id-at-authorityRevocationList }

certificateRevocationList   ATTRIBUTE   ::=     {
        WITH SYNTAX         CertificateList
        ID                  id-at-certificateRevocationList }

crossCertificatePair    ATTRIBUTE   ::=     {
        WITH SYNTAX         CertificatePair
        ID                  id-at-crossCertificatePair }

-- information object classes --

ALGORITHM   ::=     TYPE-IDENTIFIER

-- Parameterized Types --
HASHED {ToBeHashed}         ::=     OCTET STRING ( CONSTRAINED-BY {
        --must be the result of applying a hashing procedure to the --
        --DER-encoded octets of a value of -- ToBeHashed })

ENCRYPTED { To\BeEnciphered}        :=      BIT STRING ( CONSTRAINED BY {
```

```
        --must be the result of applying an encipherment  procedure to the --
        --BER-encoded octets of a value of -- ToBeEnciphered })

SIGNED { ToBeSigned }       ::=     SEQUENCE{
        ToBeSigned,
        COMPONENTS OF SIGNATURE { ToBeSigned }),

SIGNATURE { OfSignature }   ::=     SEQUENCE {
        AlgorithmIdentifier,
        ENCRYPTED { HASHED { OfSignature }}}

-- object identifier assignments --

id-at-userPassword                  OBJECT IDENTIFIER  ::=    {id-at 35}
id-at-userCertificate               OBJECT IDENTIFIER  ::=    {id-at 36}
id-at-cAcertificate                 OBJECT IDENTIFIER  ::=    {id-at 37}
id-at-authorityRevocationList       OBJECT IDENTIFIER  ::=    {id-at 38}
id-at-certificateRevocationList     OBJECT IDENTIFIER  ::=    {id-at 39}
id-at-crossCertificatePair          OBJECT IDENTIFIER  ::=    {id-at 40}
id-at-supportedAlgorithms           OBJECT IDENTIFIER  ::=    {id-at 52}
id-at-deltaRevocationList           OBJECT IDENTIFIER  ::=    {id-at 53}

END
```

Appendix B - Certificate and CRL Extensions ASN.1

```
CertificateExtensions {joint-iso-ccitt ds(5) module(1) certificateExtensions(26) 0}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL --

IMPORTS
        id-at, id-ce, id-mr, informationFramework, authenticationFramework,
                selectedAttributeTypes, upperBounds
                FROM UsefulDefinitions {joint-iso-ccitt ds(5) module(1)
                usefulDefinitions(0) 2}
        Name, RelativeDistinguishedName, ATTRIBUTE, Attribute,
                MATCHING-RULE FROM InformationFramework informationFramework
        CertificateSerialNumber, CertificateList, AlgorithmIdentifier,
                EXTENSION
                FROM AuthenticationFramework authenticationFramework
        DirectoryString
                FROM SelectedAttributeTypes selectedAttributeTypes
        ub-name
                FROM UpperBounds upperBounds
        ORAddress
                FROM MTSAbstractService {joint-iso-ccitt mhs(6) mts(3)
                modules(0) mts-abstract-service(1) version-1994 (0) } ;

-- Unless explicitly noted otherwise, there is no significance to the ordering
-- of components of a SEQUENCE OF construct in this specification.

-- Key and policy information extensions --

authorityKeyIdentifier EXTENSION ::= {
        SYNTAX              AuthorityKeyIdentifier
        IDENTIFIED BY       { id-ce 35 } }

AuthorityKeyIdentifier ::= SEQUENCE {
        keyIdentifier                 [0] KeyIdentifier            OPTIONAL,
        authorityCertIssuer           [1] GeneralNames             OPTIONAL,
        authorityCertSerialNumber     [2] CertificateSerialNumber  OPTIONAL }
        ( WITH COMPONENTS             {..., authorityCertIssuer PRESENT,
                                      authorityCertSerialNumber PRESENT} |
         WITH COMPONENTS {..., authorityCertIssuer ABSENT,
                                      authorityCertSerialNumber ABSENT} )

KeyIdentifier ::= OCTET STRING

subjectKeyIdentifier EXTENSION ::= {
        SYNTAX              SubjectKeyIdentifier
        IDENTIFIED BY       { id-ce 14 } }

SubjectKeyIdentifier ::= KeyIdentifier

keyUsage EXTENSION ::= {
```

```
        SYNTAX          KeyUsage
        IDENTIFIED BY { id-ce 15 } }

KeyUsage ::= BIT STRING {
        digitalSignature                (0),
        nonRepudiation                  (1),
        keyEncipherment                 (2),
        dataEncipherment                (3),
        keyAgreement                    (4),
        keyCertSign                     (5),
        cRLSign                         (6) }

privateKeyUsagePeriod EXTENSION ::= {
        SYNTAX          PrivateKeyUsagePeriod
        IDENTIFIED BY { id-ce 16 } }

PrivateKeyUsagePeriod ::= SEQUENCE {
        notBefore       [0]     GeneralizedTime OPTIONAL,
        notAfter        [1]     GeneralizedTime OPTIONAL }
        ( WITH COMPONENTS       {..., notBefore PRESENT} |
        WITH COMPONENTS  {..., notAfter PRESENT} )

certificatePolicies EXTENSION ::= {
        SYNTAX          CertificatePoliciesSyntax
        IDENTIFIED BY { id-ce 32 } }

CertificatePoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF PolicyInformation

PolicyInformation ::= SEQUENCE {
        policyIdentifier   CertPolicyId,
        policyQualifiers   SEQUENCE SIZE (1..MAX) OF
                PolicyQualifierInfo OPTIONAL }

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
        policyQualifierId       CERT-POLICY-QUALIFIER.&id
                                        ({SupportedPolicyQualifiers}),
        qualifier               CERT-POLICY-QUALIFIER.&Qualifier
                                        ({SupportedPolicyQualifiers}{@policyQualifierId})
                                        OPTIONAL }

SupportedPolicyQualifiers CERT-POLICY-QUALIFIER ::= { ... }

CERT-POLICY-QUALIFIER ::= CLASS {
        &id             OBJECT IDENTIFIER UNIQUE,
        &Qualifier      OPTIONAL }
WITH SYNTAX {
        POLICY-QUALIFIER-ID             &id
        [QUALIFIER-TYPE         &Qualifier] }

policyMappings EXTENSION ::= {
        SYNTAX          PolicyMappingsSyntax
        IDENTIFIED BY { id-ce 33 } }
```

```
PolicyMappingsSyntax ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        issuerDomainPolicy          CertPolicyId,
        subjectDomainPolicy  CertPolicyId }

supportedAlgorithms ATTRIBUTE ::= {
        WITH SYNTAX SupportedAlgorithm
        EQUALITY MATCHING RULE algorithmIdentifierMatch
        ID { id-at 52 } }

SupportedAlgorithm ::= SEQUENCE {
        algorithmIdentifier          AlgorithmIdentifier,
        intendedUsage                [0]  KeyUsage OPTIONAL,
        intendedCertificatePolicies  [1]  CertificatePoliciesSyntax OPTIONAL }

-- Certificate subject and certificate issuer attributes extensions --

subjectAltName EXTENSION ::= {
        SYNTAX        GeneralNames
        IDENTIFIED BY { id-ce 17 } }

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
        otherName                    [0]    INSTANCE OF OTHER-NAME,
        rfc822Name                   [1]    IA5String,
        dNSName                      [2]    IA5String,
        x400Address                  [3]    ORAddress,
        directoryName                [4]    Name,
        ediPartyName                 [5]    EDIPartyName,
        uniformResourceIdentifier    [6]    IA5String,
        iPAddress                    [7]    OCTET STRING,
        registeredID                 [8]    OBJECT IDENTIFIER }

OTHER-NAME ::= TYPE-IDENTIFIER

EDIPartyName ::= SEQUENCE {
        nameAssigner                 [0]    DirectoryString {ub-name} OPTIONAL,
        partyName                    [1]    DirectoryString {ub-name} }

issuerAltName EXTENSION ::= {
        SYNTAX        GeneralNames
        IDENTIFIED BY { id-ce 18 } }

subjectDirectoryAttributes EXTENSION ::= {
        SYNTAX        AttributesSyntax
        IDENTIFIED BY { id-ce 9 } }

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


-- Certification path constraints extensions --

basicConstraints EXTENSION ::= {
```

```
        SYNTAX          BasicConstraintsSyntax
        IDENTIFIED BY { id-ce 19 } }

BasicConstraintsSyntax ::= SEQUENCE {
        cA                          BOOLEAN DEFAULT FALSE,
        pathLenConstraint      INTEGER (0..MAX) OPTIONAL }

nameConstraints EXTENSION ::= {
        SYNTAX          NameConstraintsSyntax
        IDENTIFIED BY { id-ce 30 } }

NameConstraintsSyntax ::= SEQUENCE {
        permittedSubtrees      [0]      GeneralSubtrees OPTIONAL,
        excludedSubtrees       [1]      GeneralSubtrees OPTIONAL }

GeneralSubtrees ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree

GeneralSubtree ::= SEQUENCE {
        base                        GeneralName,
        minimum                     [0]      BaseDistance DEFAULT 0,
        maximum                     [1]      BaseDistance OPTIONAL }

BaseDistance ::= INTEGER (0..MAX)

policyConstraints EXTENSION ::= {
        SYNTAX          PolicyConstraintsSyntax
        IDENTIFIED BY { id-ce 36 } }

PolicyConstraints Syntax ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        requireExplicitPolicy   [0] SkipCerts OPTIONAL,
        inhibitPolicyMapping   [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)

-- Basic CRL extensions --

cRLNumber EXTENSION ::= {
        SYNTAX          CRLNumber
        IDENTIFIED BY { id-ce 20 } }

CRLNumber ::= INTEGER (0..MAX)

reasonCode EXTENSION ::= {
        SYNTAX          CRLReason
        IDENTIFIED BY { id-ce 21 } }

CRLReason ::= ENUMERATED {
        unspecified                     (0),
        keyCompromise                   (1),
        cACompromise                    (2),
        affiliationChanged              (3),
        superseded                      (4),
        cessationOfOperation            (5),
        certificateHold                 (6),
```

```
        removeFromCRL                   (8) }

instructionCode EXTENSION ::= {
        SYNTAX          HoldInstruction
        IDENTIFIED BY { id-ce 23 } }

HoldInstruction ::= OBJECT IDENTIFIER

invalidityDate EXTENSION ::= {
        SYNTAX          GeneralizedTime
        IDENTIFIED BY { id-ce 24 } }


-- CRL distribution points and delta-CRL extensions --

cRLDistributionPoints EXTENSION ::= {
        SYNTAX          CRLDistPointsSyntax
        IDENTIFIED BY           { id-ce 31 } }

CRLDistPointsSyntax ::= SEQUENCE SIZE (1..MAX) OF DistributionPoint

DistributionPoint ::= SEQUENCE {
        distributionPoint       [0]     DistributionPointName OPTIONAL,
        reasons                 [1]     ReasonFlags OPTIONAL,
        cRLIssuer               [2]     GeneralNames OPTIONAL }

DistributionPointName ::= CHOICE {
        fullName                        [0]     GeneralNames,
        nameRelativeToCRLIssuer         [1]     RelativeDistinguishedName }

ReasonFlags ::= BIT STRING {
        unused                          (0),
        keyCompromise                   (1),
        caCompromise                    (2),
        affiliationChanged              (3),
        superseded                      (4),
        cessationOfOperation            (5),
        certificateHold                 (6) }

issuingDistributionPoint EXTENSION ::= {
        SYNTAX          IssuingDistPointSyntax
        IDENTIFIED BY           { id-ce 28 } }

IssuingDistPointSyntax ::= SEQUENCE {
        distributionPoint               [0] DistributionPointName OPTIONAL,
        onlyContainsUserCerts           [1] BOOLEAN DEFAULT FALSE,
        onlyContainsCACerts             [2] BOOLEAN DEFAULT FALSE,
        onlySomeReasons                 [3] ReasonFlags OPTIONAL,
        indirectCRL                     [4] BOOLEAN DEFAULT FALSE }

certificateIssuer EXTENSION ::= {
        SYNTAX                  GeneralNames
        IDENTIFIED BY           { id-ce 29 } }
```

```
deltaCRLIndicator EXTENSION ::= {
        SYNTAX                  BaseCRLNumber
        IDENTIFIED BY           { id-ce 27 } }

BaseCRLNumber ::= CRLNumber

deltaRevocationList ATTRIBUTE ::= {
        WITH SYNTAX CertificateList
        EQUALITY MATCHING RULE certificateListExactMatch
        ID      {id-at 53 } }


-- Matching rules --

certificateExactMatch MATCHING-RULE ::= {
        SYNTAX                  CertificateExactAssertion
        ID                      id-mr-certificateExactMatch }

CertificateExactAssertion ::= SEQUENCE {
        serialNumber            CertificateSerialNumber,
        issuer                  Name }

certificateMatch MATCHING-RULE ::= {
        SYNTAX                  CertificateAssertion
        ID                      id-mr-certificateMatch }

CertificateAssertion ::= SEQUENCE {
        serialNumber                [0] CertificateSerialNumber OPTIONAL,
        issuer                      [1] Name                    OPTIONAL,
        subjectKeyIdentifier        [2] SubjectKeyIdentifier    OPTIONAL,
        authorityKeyIdentifier      [3] AuthorityKeyIdentifier  OPTIONAL,
        certificateValid            [4] UTCTime                 OPTIONAL,
        privateKeyValid             [5] GeneralizedTime         OPTIONAL,
        subjectPublicKeyAlgID       [6] OBJECT IDENTIFIER       OPTIONAL,
        keyUsage                    [7] KeyUsage                OPTIONAL,
        subjectAltName              [8] AltNameType             OPTIONAL,
        policy                      [9] CertPolicySet           OPTIONAL,
        pathToName                  [10] Name                   OPTIONAL }

AltNameType ::= CHOICE {
        builtinNameForm         ENUMERATED {
                                    rfc822Name              (1),
                                    dNSName                 (2),
                                    x400Address             (3),
                                    directoryName (4),
                                    ediPartyName  (5),
                                    uniformResourceIdentifier       (6),
                                    iPAddress               (7),
                                    registeredId            (8) },
        otherNameForm           OBJECT IDENTIFIER }
```

```
certificatePairExactMatch MATCHING-RULE  ::=      {
        SYNTAX                  CertificatePairExactAssertion
        ID                      id-mr-certificatePairExactMatch }

CertificatePairExactAssertion ::= SEQUENCE {
        forwardAssertion        [0] CertificateExactAssertion OPTIONAL,
        reverseAssertion        [1] CertificateExactAssertion OPTIONAL }
        ( WITH COMPONENTS           {..., forwardAssertion PRESENT} |
          WITH COMPONENTS           {..., reverseAssertion PRESENT} )

certificatePairMatch MATCHING-RULE ::=      {
        SYNTAX          CertificatePairAssertion
        ID              id-mr-certificatePairMatch }

CertificatePairAssertion ::= SEQUENCE {
        forwardAssertion        [0] CertificateAssertion OPTIONAL,
        reverseAssertion        [1] CertificateAssertion OPTIONAL }
        ( WITH COMPONENTS           {..., forwardAssertion PRESENT} |
          WITH COMPONENTS           {..., reverseAssertion PRESENT} )

certificateListExactMatch MATCHING-RULE  ::=      {
        SYNTAX                  CertificateListExactAssertion
        ID                      id-mr-certificateListExactMatch }

CertificateListExactAssertion ::= SEQUENCE {
        issuer                  Name,
        thisUpdate              UTCTime,
        distributionPoint       DistributionPointName OPTIONAL }

certificateListMatch MATCHING-RULE ::= {
        SYNTAX                  CertificateListAssertion
        ID                      id-mr-certificateListMatch }

CertificateListAssertion ::= SEQUENCE {
        issuer                  Name   OPTIONAL,
        minCRLNumber            [0]     CRLNumber    OPTIONAL,
        maxCRLNumber            [1]     CRLNumber    OPTIONAL,
        reasonFlags             ReasonFlags   OPTIONAL,
        dateAndTime             UTCTime       OPTIONAL,
        distributionPoint       [2]     DistributionPointName OPTIONAL }

algorithmIdentifierMatch MATCHING-RULE   ::=      {
        SYNTAX                  AlgorithmIdentifier
        ID                      id-mr-algorithmIdentifierMatch }


-- Object identifier assignments --

id-at-supportedAlgorithms           OBJECT IDENTIFIER  ::=     {id-at 52}
id-at-deltaRevocationList           OBJECT IDENTIFIER  ::=     {id-at 53}
id-ce-subjectDirectoryAttributes    OBJECT IDENTIFIER  ::=     {id-ce 9}
id-ce-subjectKeyIdentifier          OBJECT IDENTIFIER  ::=     {id-ce 14}
id-ce-keyUsage                      OBJECT IDENTIFIER  ::=     {id-ce 15}
id-ce-privateKeyUsagePeriod         OBJECT IDENTIFIER  ::=     {id-ce 16}
```

```
id-ce-subjectAltName              OBJECT IDENTIFIER  ::=    {id-ce 17}
id-ce-issuerAltName               OBJECT IDENTIFIER  ::=    {id-ce 18}
id-ce-basicConstraints            OBJECT IDENTIFIER  ::=    {id-ce 19}
id-ce-cRLNumber                   OBJECT IDENTIFIER  ::=    {id-ce 20}
id-ce-reasonCode                  OBJECT IDENTIFIER  ::=    {id-ce 21}
id-ce-instructionCode             OBJECT IDENTIFIER  ::=    {id-ce 23}
id-ce-invalidityDate              OBJECT IDENTIFIER  ::=    {id-ce 24}
id-ce-deltaCRLIndicator           OBJECT IDENTIFIER  ::=    {id-ce 27}
id-ce-issuingDistributionPoint    OBJECT IDENTIFIER  ::=    {id-ce 28}
id-ce-certificateIssuer           OBJECT IDENTIFIER  ::=    {id-ce 29}
id-ce-nameConstraints             OBJECT IDENTIFIER  ::=    {id-ce 30}
id-ce-cRLDistributionPoints       OBJECT IDENTIFIER  ::=    {id-ce 31}
id-ce-certificatePolicies         OBJECT IDENTIFIER  ::=    {id-ce 32}
id-ce-policyMappings              OBJECT IDENTIFIER  ::=    {id-ce 33}
id-ce-policyConstraints           OBJECT IDENTIFIER  ::=    {id-ce 34}
id-ce-authorityKeyIdentifier      OBJECT IDENTIFIER  ::=    {id-ce 35}
id-mr-certificateExactMatch       OBJECT IDENTIFIER  ::=    {id-mr 34}
id-mr-certificateMatch            OBJECT IDENTIFIER  ::=    {id-mr 35}
id-mr-certificatePairExactMatch   OBJECT IDENTIFIER  ::=    {id-mr 36}
id-mr-certificatePairMatch        OBJECT IDENTIFIER  ::=    {id-mr 37}
id-mr-certificateListExactMatch   OBJECT IDENTIFIER  ::=    {id-mr 38}
id-mr-certificateListMatch        OBJECT IDENTIFIER  ::=    {id-mr 39}
id-mr-algorithmIdentifierMatch    OBJECT IDENTIFIER  ::=    {id-mr 40}

-- The following OBJECT IDENTIFIERS are not used by this specification:
-- {id-ce 2}, {id-ce 3}, {id-ce 4}, {id-ce 5}, {id-ce 6}, {id-ce 7},
-- {id-ce 8}, {id-ce 10}, {id-ce 11}, {id-ce 12}, {id-ce 13},
-- {id-ce 22}, {id-ce 25}, {id-ce 26}

END
```

Appendix C - ASN.1 Module for transactions

The following section contains the complete ASN.1 module from PKIX Part 3. Only a small subset of the messages defined in PKIX Part 3 are required to implement this specification. The entire module is provided for completeness. Information about messages defined by this ASN.1 module but not used in the MISPC may be found in [PKIX3].

```
PKIX3
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

PKIMessage ::= SEQUENCE {
    header          PKIHeader,
    body            PKIBody,
    protection   [0] PKIProtection OPTIONAL,
    extraCerts   [1] SEQUENCE OF Certificate OPTIONAL
 }

 PKIHeader ::= SEQUENCE {
    pvno                 INTEGER     { ietf-version1 (0) },
    sender               GeneralName,
    -- identifies the sender
    recipient            GeneralName,
    -- identifies the intended recipient
    messageTime          [0] GeneralizedTime       OPTIONAL,
    -- time of production of this message (used when sender
    -- believes that the transport will be "suitable"; i.e.,
    -- that the time will still be meaningful upon receipt)
    protectionAlg        [1] AlgorithmIdentifier   OPTIONAL,
    -- algorithm used for calculation of protection bits
    senderKID            [2] KeyIdentifier         OPTIONAL,
    recipKID             [3] KeyIdentifier         OPTIONAL,
    -- to identify specific keys used for protection
    transactionID        [4] OCTET STRING          OPTIONAL,
    -- identifies the transaction, i.e., this will be the same in
    -- corresponding request, response and confirmation messages
    senderNonce          [5] OCTET STRING          OPTIONAL,
    recipNonce           [6] OCTET STRING          OPTIONAL,
    -- nonces used to provide replay protection, senderNonce
    -- is inserted by the creator of this message; recipNonce
    -- is a nonce previously inserted in a related message by
    -- the intended recipient of this message
    freeText             [7] PKIFreeText          OPTIONAL
    -- this may be used to indicate context-specific
    -- instructions (this field is intended for human
    -- consumption)
 }

 PKIFreeText ::= CHOICE {
    iA5String   [0] IA5String,
    bMPString  [1] BMPString
 }
```

```
PKIBody ::= CHOICE {       -- message-specific body elements
   ir             [0]  InitReqContent,
   ip             [1]  InitRepContent,
   cr             [2]  CertReqContent,
   cp             [3]  CertRepContent,
   p10cr          [4]  PKCS10CertReqContent, -- imported from [PKCS10]
   popdecc        [5]  POPODecKeyChallContent,
   popdecr        [6]  POPODecKeyRespContent,
   kur            [7]  KeyUpdReqContent,
   kup            [8]  KeyUpdRepContent,
   krr            [9]  KeyRecReqContent,
   krp            [10] KeyRecRepContent,
   rr             [11] RevReqContent,
   rp             [12] RevRepContent,
   ccr            [13] CrossCertReqContent,
   ccp            [14] CrossCertRepContent,
   ckuann         [15] CAKeyUpdAnnContent,
   cann           [16] CertAnnContent,
   rann           [17] RevAnnContent,
   crlann         [18] CRLAnnContent,
   conf           [19] PKIConfirmContent,
   nested         [20] NestedMessageContent,
   infor          [21] PKIInfoReqContent,
   infop          [22] PKIInfoRepContent,
   error          [23] ErrorMsgContent
}

PKIProtection ::= BIT STRING

ProtectedPart ::= SEQUENCE {
   header    PKIHeader,
   body      PKIBody
}

PasswordBasedMac ::= OBJECT IDENTIFIER

PBMParameter ::= SEQUENCE {
   salt              OCTET STRING,
   owf               AlgorithmIdentifier,
   -- AlgId for a One-Way Function (SHA-1 recommended)
   iterationCount    INTEGER,
   -- number of times the OWF is applied
   mac               AlgorithmIdentifier
   -- the MAC AlgId (e.g., DES-MAC or Triple-DES-MAC [PKCS #11])
}

DHBasedMac ::= OBJECT IDENTIFIER

DHBMParameter ::= SEQUENCE {
   owf               AlgorithmIdentifier,
   -- AlgId for a One-Way Function (SHA-1 recommended)
   mac               AlgorithmIdentifier
   -- the MAC AlgId (e.g., DES-MAC or Triple-DES-MAC [PKCS #11])
```

```
}

NestedMessageContent ::= ANY
-- This will be a PKIMessage


CertTemplate ::= SEQUENCE {
   version      [0] Version             OPTIONAL,
   -- used to ask for a particular syntax version
   serial       [1] INTEGER             OPTIONAL,
   -- used to ask for a particular serial number
   signingAlg          [2] AlgorithmIdentifier   OPTIONAL,
   -- used to ask the CA to use this alg. for signing the cert
   subject      [3] Name                OPTIONAL,
   validity     [4] OptionalValidity     OPTIONAL,
   issuer       [5] Name                OPTIONAL,
   publicKey  [6] SubjectPublicKeyInfo  OPTIONAL,
   issuerUID  [7] UniqueIdentifier      OPTIONAL,
   subjectUID [8] UniqueIdentifier      OPTIONAL,
   extensions [9] Extensions           OPTIONAL
   -- the extensions which the requester would like in the cert.
}

OptionalValidity ::= SEQUENCE {
   notBefore  [0] UTCTime OPTIONAL,
   notAfter   [1] UTCTime OPTIONAL
}

EncryptedValue ::= SEQUENCE {
   encValue             BIT STRING,
   -- the encrypted value itself
   intendedAlg          [0] AlgorithmIdentifier  OPTIONAL,
   -- the intended algorithm for which the value will be used
   symmAlg              [1] AlgorithmIdentifier  OPTIONAL,
   -- the symmetric algorithm used to encrypt the value
   encSymmKey        [2] BIT STRING          OPTIONAL,
   -- the (encrypted) symmetric key used to encrypt the value
   keyAlg                          [3] AlgorithmIdentifier  OPTIONAL
   -- algorithm used to encrypt the symmetric key
}

PKIStatus ::= INTEGER {
   granted                        (0),
   -- you got exactly what you asked for
   grantedWithMods            (1),
   -- you got something like what you asked for; the
   -- requester is responsible for ascertaining the differences
   rejection                      (2),
   -- you don't get it, more information elsewhere in the message
   waiting                        (3),
   -- the request body part has not yet been processed,
   -- expect to hear more later
   revocationWarning                     (4),
   -- this message contains a warning that a revocation is
```

```
    -- imminent
    revocationNotification       (5),
    -- notification that a revocation has occurred
    keyUpdateWarning             (6)
    -- update already done for the oldCertId specified in
    -- FullCertTemplate
}

PKIFailureInfo ::= BIT STRING {
-- since we can fail in more than one way!
        badAlg                   (0),     -- unrecognized  or unsupported algorithm identifier
        badMessageCheck          (1),     -- integrity check failed (e.g., signature did not verify)
        badRequest               (2),     -- transaction not permitted or supported
        badTime                  (3),     -- messageTime field was not sufficiently close
                                          -- to the system time, as defined by local policy
        badCertId                (4),     -- no certificate could be identified matching the
                                          -- provided criteria
        badPoP                   (5)      -- proof of possession field did not verify
    -- more TBS
}

PKIStatusInfo ::= SEQUENCE {
    status        PKIStatus,
    statusString  PKIFreeText     OPTIONAL,
    failInfo      PKIFailureInfo  OPTIONAL
}

CertId ::= SEQUENCE {
    issuer          GeneralName,
    serialNumber    INTEGER
}

OOBCert ::= Certificate

OOBCertHash ::= SEQUENCE {
    hashAlg     [0] AlgorithmIdentifier     OPTIONAL,
    certId      [1] CertId                  OPTIONAL,
    hashVal         BIT STRING
    -- hashVal is calculated over DER encoding of the
    -- subjectPublicKey field of the corresponding cert.
}

PKIArchiveOptions ::= CHOICE {
    encryptedPrivKey     [0] EncryptedValue,
    -- the actual value of the private key
    keyGenParameters     [1] KeyGenParameters,
    -- parameters which allow the private key to be re-generated
    archiveRemGenPrivKey [2] BOOLEAN
    -- set to TRUE if sender wishes receiver to archive the private
    -- key of a key pair which the receiver generates in response to
    -- this request; set to FALSE if no archival is desired.
}

KeyGenParameters ::= OCTET STRING
```

```
   -- actual syntax is <<TBS>>
   -- an alternative to sending the key is to send the information
   -- about how to re-generate the key (e.g. for many RSA
   -- implementations one could send the first random number tested
   -- for primality)

PKIPublicationInfo ::= SEQUENCE {
  action     INTEGER {
             dontPublish (0),
             pleasePublish (1)
            },
  pubInfos  SEQUENCE OF SinglePubInfo OPTIONAL
    -- pubInfos must not be present if action is "dontPublish"
    -- (if action is "pleasePublish" and pubInfos is omitted,
    -- "dontCare" is assumed)
}

SinglePubInfo ::= SEQUENCE {
   pubMethod    INTEGER {
      dontCare    (0),
      x500        (1),
      web         (2)
   },
   pubLocation  GeneralName OPTIONAL
}

FullCertTemplates ::= SEQUENCE OF FullCertTemplate

FullCertTemplate ::= SEQUENCE {
   certReqId             INTEGER,
   -- to match this request with corresponding response
   -- (note:  must be unique over all FullCertReqs in this message)
   certTemplate          CertTemplate,
   popoSigningKey     [0] POPOSigningKey      OPTIONAL,
   archiveOptions     [1] PKIArchiveOptions   OPTIONAL,
   publicationInfo    [2] PKIPublicationInfo  OPTIONAL,
   oldCertId          [3] CertId              OPTIONAL
   -- id. of cert. which is being updated by this one
}

POPOSigningKey ::= SEQUENCE {
   poposkInput         POPOSKInput,
   alg                 AlgorithmIdentifier,
   signature           BIT STRING
   -- the signature (using "alg") on the DER-encoded
   -- value of poposkInput
}

POPOSKInput ::= CHOICE {
   popoSigningKeyInput      [0] POPOSigningKeyInput,
   certificationRequestInfo     CertificationRequestInfo
   -- imported from [PKCS10] (note that if this choice is used,
   -- POPOSigningKey is simply a standard PKCS #10 request; this
   -- allows a bare PKCS #10 request to be augmented with other
```

```
   -- desired information in the FullCertTemplate before being
   -- sent to the CA/RA)
}

POPOSigningKeyInput ::= SEQUENCE {
   authInfo            CHOICE {
      sender             [0] GeneralName,
      -- from PKIHeader (used only if an authenticated identity
      -- has been established for the sender (e.g., a DN from a
      -- previously-issued and currently-valid certificate)
      publicKeyMAC       [1] BIT STRING
      -- used if no authenticated GeneralName currently exists for
      -- the sender; publicKeyMAC contains a password-based MAC
      -- (using the protectionAlg AlgId from PKIHeader) on the
      -- DER-encoded value of publicKey
   },
   publicKey           SubjectPublicKeyInfo    -- from CertTemplate
}

InitReqContent ::= SEQUENCE {
   protocolEncKey      [0] SubjectPublicKeyInfo  OPTIONAL,
   fullCertTemplates       FullCertTemplates
}

InitRepContent ::= CertRepContent

CertReqContent ::= CHOICE {
   fullCertTemplates    [0] FullCertTemplates,
   pkcs10CertReqContent [1] PKCS10CertReqContent
}

POPODecKeyChallContent ::= SEQUENCE OF Challenge
-- One Challenge per encryption key certification request (in the
-- same order as these requests appear in FullCertTemplates).

Challenge ::= SEQUENCE {
   owf                AlgorithmIdentifier  OPTIONAL,
   -- must be present in the first Challenge; may be omitted in any
   -- subsequent Challenge in POPODecKeyChallContent (if omitted,
   -- then the owf used in the immediately preceding Challenge is
   -- to be used).
   witness            OCTET STRING,
   -- the result of applying the one-way function (owf) to a
   -- randomly-generated INTEGER, A.  [Note that a different
   -- INTEGER must be used for each Challenge.]
   challenge          OCTET STRING
   -- the encryption (under the public key for which the cert.
   -- request is being made) of Rand, where Rand is specified as
   --   Rand ::= SEQUENCE {
   --      int      INTEGER,
   --       - the randomly-generated INTEGER A (above)
   --      sender   GeneralName
   --       - the sender's name (as included in PKIHeader)
   --   }
```

```
}

POPODecKeyRespContent ::= SEQUENCE OF INTEGER
-- One INTEGER per encryption key certification request (in the
-- same order as these requests appear in FullCertTemplates).  The
-- retrieved INTEGER A (above) is returned to the sender of the
-- corresponding Challenge.

CertRepContent ::= SEQUENCE {
   caPub          [1] Certificate          OPTIONAL,
   response           SEQUENCE OF CertResponse
}

CertResponse ::= SEQUENCE {
   certReqId          INTEGER,
   -- to match this response with corresponding request
   status             PKIStatusInfo,
   certifiedKeyPair   CertifiedKeyPair   OPTIONAL
}

CertifiedKeyPair ::= SEQUENCE {
   certificate    [0] Certificate        OPTIONAL,
   encryptedCert  [1] EncryptedValue     OPTIONAL,
   privateKey     [2] EncryptedValue     OPTIONAL,
   publicationInfo [3] PKIPublicationInfo  OPTIONAL
}

KeyUpdReqContent ::= SEQUENCE {
   protocolEncKey     [0] SubjectPublicKeyInfo  OPTIONAL,
   fullCertTemplates  [1] FullCertTemplates     OPTIONAL
}

KeyUpdRepContent ::= InitRepContent

KeyRecReqContent ::= InitReqContent

KeyRecRepContent ::= SEQUENCE {
   status                PKIStatusInfo,
   newSigCert         [0] Certificate                 OPTIONAL,
   caCerts            [1] SEQUENCE OF Certificate      OPTIONAL,
   keyPairHist        [2] SEQUENCE OF CertifiedKeyPair  OPTIONAL
}

RevReqContent ::= SEQUENCE OF RevDetails

RevDetails ::= SEQUENCE {
   certDetails        CertTemplate,
   -- allows requester to specify as much as they can about
   -- the cert. for which revocation is requested
   -- (e.g. for cases in which serialNumber is not available)
   revocationReason    ReasonFlags,
   -- from the DAM, so that CA knows which Dist. point to use
   badSinceDate        GeneralizedTime  OPTIONAL,
   -- indicates best knowledge of sender
```

```
    crlEntryDetails     Extensions
    -- requested crlEntryExtensions
  }

  RevRepContent ::= SEQUENCE {
    status              PKIStatusInfo,
    revCerts        [0] SEQUENCE OF CertId OPTIONAL,
    -- identifies the certs for which revocation was requested
    crls            [1] SEQUENCE OF CertificateList  OPTIONAL
    -- the resulting CRLs (there may be more than one)
  }

  CrossCertReqContent ::= CertReqContent

  CrossCertRepContent ::= CertRepContent

  CAKeyUpdAnnContent ::= SEQUENCE {
    oldWithNew          Certificate, -- old pub signed with new priv
    newWithOld          Certificate, -- new pub signed with old priv
    newWithNew          Certificate  -- new pub signed with new priv
  }

  CertAnnContent ::= Certificate

  RevAnnContent ::= SEQUENCE {
    status              PKIStatus,
    certId              CertId,
    willBeRevokedAt     GeneralizedTime,
    badSinceDate        GeneralizedTime,
    crlDetails          Extensions  OPTIONAL
    -- extra CRL details(e.g., crl number, reason, location, etc.)
}

  CRLAnnContent ::= SEQUENCE OF CertificateList

  PKIConfirmContent ::= NULL

  InfoTypeAndValue ::= SEQUENCE {
    infoType            OBJECT IDENTIFIER,
    infoValue           ANY DEFINED BY infoType  OPTIONAL
  }
  -- Example InfoTypeAndValue contents include, but are not limited to:
  --   { CAProtEncCert    = { xx }, Certificate                      }
  --   { SignKeyPairTypes = { xx }, SEQUENCE OF AlgorithmIdentifier }
  --   { EncKeyPairTypes  = { xx }, SEQUENCE OF AlgorithmIdentifier }
  --   { PreferredSymmAlg = { xx }, AlgorithmIdentifier              }
  --   { CAKeyUpdateInfo  = { xx }, CAKeyUpdAnnContent               }
  --   { CurrentCRL       = { xx }, CertificateList                  }


  PKIInfoReqContent ::= SET OF InfoTypeAndValue
  -- The OPTIONAL infoValue parameter of InfoTypeAndValue is unused.
  -- The CA is free to ignore any contained OBJ. IDs that it does not
  -- recognize.
```

```
-- The empty set indicates that the CA may send any/all information
-- that it wishes.

PKIInfoRepContent ::= SET OF InfoTypeAndValue
-- The end-entity is free to ignore any contained OBJ. IDs that it
-- does not recognize.

ErrorMsgContent ::= SEQUENCE {
   pKIStatusInfo          PKIStatusInfo,
   errorCode              INTEGER           OPTIONAL,
   -- implementation-specific error codes
   errorDetails           PKIFreeText       OPTIONAL
   -- implementation-specific error details
}
```