Network Working Group
Request for Comments: 3280
Obsoletes: 2459
Category: Standards Track

R. Housley
RSA Laboratories
W. Polk
NIST
W. Ford
VeriSign
D. Solo
Citigroup

April 2002

Internet X.509 Public Key Infrastructure
Certificate and Certificate Revocation List (CRL) Profile

Status of this Memo

Copyright Notice

Abstract

This memo profiles the X.509 v3 certificate and X.509 v2 Certificate Revocation List (CRL) for use in the Internet. An overview of this approach and model are provided as an introduction. The X.509 v3 certificate format is described in detail, with additional information regarding the format and semantics of Internet name forms. Standard certificate extensions are described and two Internet-specific extensions are defined. A set of required certificate extensions is specified. The X.509 v2 CRL format is described in detail, and required extensions are defined. An algorithm for X.509 certification path validation is described. An ASN.1 module and examples are provided in the appendices.

Table of Contents

1 Introduction

This specification is one part of a family of standards for the X.509 Public Key Infrastructure (PKI) for the Internet.

This specification profiles the format and semantics of certificates and certificate revocation lists (CRLs) for the Internet PKI. Procedures are described for processing of certification paths in the Internet environment. Finally, ASN.1 modules are provided in the appendices for all data structures defined or referenced.

Section 2 describes Internet PKI requirements, and the assumptions which affect the scope of this document. Section 3 presents an architectural model and describes its relationship to previous IETF and ISO/IEC/ITU-T standards. In particular, this document's relationship with the IETF PEM specifications and the ISO/IEC/ITU-T X.509 documents are described.

Section 4 profiles the X.509 version 3 certificate, and section 5 profiles the X.509 version 2 CRL. The profiles include the identification of ISO/IEC/ITU-T and ANSI extensions which may be useful in the Internet PKI. The profiles are presented in the 1988 Abstract Syntax Notation One (ASN.1) rather than the 1997 ASN.1 syntax used in the most recent ISO/IEC/ITU-T standards.

Section 6 includes certification path validation procedures. These procedures are based upon the ISO/IEC/ITU-T definition. Implementations are REQUIRED to derive the same results but are not required to use the specified procedures.

Procedures for identification and encoding of public key materials and digital signatures are defined in [PKIXALGS]. Implementations of this specification are not required to use any particular cryptographic algorithms. However, conforming implementations which use the algorithms identified in [PKIXALGS] MUST identify and encode the public key materials and digital signatures as described in that specification.

Finally, three appendices are provided to aid implementers. Appendix A contains all ASN.1 structures defined or referenced within this specification. As above, the material is presented in the 1988 ASN.1. Appendix B contains notes on less familiar features of the ASN.1 notation used within this specification. Appendix C contains examples of a conforming certificate and a conforming CRL.

This specification obsoletes RFC 2459. This specification differs from RFC 2459 in five basic areas:

* To promote interoperable implementations, a detailed algorithm for certification path validation is included in section 6.1 of this specification; RFC 2459 provided only a high-level description of path validation.

* An algorithm for determining the status of a certificate using CRLs is provided in section 6.3 of this specification. This material was not present in RFC 2459.

* To accommodate new usage models, detailed information describing the use of delta CRLs is provided in Section 5 of this specification.

* Identification and encoding of public key materials and digital signatures are not included in this specification, but are now described in a companion specification [PKIXALGS].

* Four additional extensions are specified: three certificate extensions and one CRL extension. The certificate extensions are subject info access, inhibit any-policy, and freshest CRL. The freshest CRL extension is also defined as a CRL extension.

* Throughout the specification, clarifications have been introduced to enhance consistency with the ITU-T X.509 specification. X.509 defines the certificate and CRL format as well as many of the extensions that appear in this specification. These changes were introduced to improve the likelihood of interoperability between implementations based on this specification with implementations based on the ITU-T specification.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2 Requirements and Assumptions

The goal of this specification is to develop a profile to facilitate the use of X.509 certificates within Internet applications for those communities wishing to make use of X.509 technology. Such applications may include WWW, electronic mail, user authentication, and IPsec. In order to relieve some of the obstacles to using X.509

certificates, this document defines a profile to promote the development of certificate management systems; development of application tools; and interoperability determined by policy.

Some communities will need to supplement, or possibly replace, this profile in order to meet the requirements of specialized application domains or environments with additional authorization, assurance, or operational requirements. However, for basic applications, common representations of frequently used attributes are defined so that application developers can obtain necessary information without regard to the issuer of a particular certificate or certificate revocation list (CRL).

A certificate user should review the certificate policy generated by the certification authority (CA) before relying on the authentication or non-repudiation services associated with the public key in a particular certificate. To this end, this standard does not prescribe legally binding rules or duties.

As supplemental authorization and attribute management tools emerge, such as attribute certificates, it may be appropriate to limit the authenticated attributes that are included in a certificate. These other management tools may provide more appropriate methods of conveying many authenticated attributes.

2.1 Communication and Topology

The users of certificates will operate in a wide range of environments with respect to their communication topology, especially users of secure electronic mail. This profile supports users without high bandwidth, real-time IP connectivity, or high connection availability. In addition, the profile allows for the presence of firewall or other filtered communication.

This profile does not assume the deployment of an X.500 Directory system or a LDAP directory system. The profile does not prohibit the use of an X.500 Directory or a LDAP directory; however, any means of distributing certificates and certificate revocation lists (CRLs) may be used.

2.2 Acceptability Criteria

The goal of the Internet Public Key Infrastructure (PKI) is to meet the needs of deterministic, automated identification, authentication, access control, and authorization functions. Support for these services determines the attributes contained in the certificate as well as the ancillary control information in the certificate such as policy data and certification path constraints.

2.3 User Expectations

Users of the Internet PKI are people and processes who use client software and are the subjects named in certificates. These uses include readers and writers of electronic mail, the clients for WWW browsers, WWW servers, and the key manager for IPsec within a router. This profile recognizes the limitations of the platforms these users employ and the limitations in sophistication and attentiveness of the users themselves. This manifests itself in minimal user configuration responsibility (e.g., trusted CA keys, rules), explicit platform usage constraints within the certificate, certification path constraints which shield the user from many malicious actions, and applications which sensibly automate validation functions.

2.4 Administrator Expectations

As with user expectations, the Internet PKI profile is structured to support the individuals who generally operate CAs. Providing administrators with unbounded choices increases the chances that a subtle CA administrator mistake will result in broad compromise. Also, unbounded choices greatly complicate the software that process and validate the certificates created by the CA.

3 Overview of Approach

Following is a simplified view of the architectural model assumed by the PKIX specifications.

The components in this model are:

end entity:	user of PKI certificates and/or end user system that is the subject of a certificate;
CA:	certification authority;
RA:	registration authority, i.e., an optional system to which a CA delegates certain management functions;
CRL issuer:	an optional system to which a CA delegates the publication of certificate revocation lists;
repository:	a system or collection of distributed systems that stores certificates and CRLs and serves as a means of distributing these certificates and CRLs to end entities.

Note that an Attribute Authority (AA) might also choose to delegate the publication of CRLs to a CRL issuer.

```
   +---+
   | C |                       +------------+
   | e | <-------------------->| End entity |
   | r |       Operational     +------------+
   | t |       transactions          ^
   | i |      and management         |  Management
   | f |       transactions          |  transactions        PKI
   | i |                             |                      users
   | c |                             v
   | a | =======================  +--+------------+  ==============
   | t |                          ^               ^
   | e |                          |               |         PKI
   |   |                          v               |      management
   | & |                       +------+           |       entities
   |   | <---------------------|  RA  |<----+     |
   | C |  Publish certificate  +------+     |     |
   | R |                                    |     |
   | L |                                    |     |
   |   |                                    v     v
   | R |                                +------------+
   | e | <------------------------------|     CA     |
   | p |   Publish certificate          +------------+
   | o |   Publish CRL                     ^      ^
   | s |                                   |      |  Management
   | i |                +------------+     |      |  transactions
   | t | <--------------| CRL Issuer |<----+      |
   | o |   Publish CRL  +------------+            v
   | r |                                      +------+
   | y |                                      |  CA  |
   +---+                                      +------+
```

Figure 1 - PKI Entities

3.1 X.509 Version 3 Certificate

Users of a public key require confidence that the associated private key is owned by the correct remote subject (person or system) with which an encryption or digital signature mechanism will be used. This confidence is obtained through the use of public key certificates, which are data structures that bind public key values to subjects. The binding is asserted by having a trusted CA digitally sign each certificate. The CA may base this assertion upon technical means (a.k.a., proof of possession through a challenge-response protocol), presentation of the private key, or on an assertion by the subject. A certificate has a limited valid lifetime which is indicated in its signed contents. Because a certificate's signature and timeliness can be independently checked by a certificate-using client, certificates can be distributed via untrusted communications and server systems, and can be cached in unsecured storage in certificate-using systems.

ITU-T X.509 (formerly CCITT X.509) or ISO/IEC 9594-8, which was first published in 1988 as part of the X.500 Directory recommendations, defines a standard certificate format [X.509]. The certificate format in the 1988 standard is called the version 1 (v1) format. When X.500 was revised in 1993, two more fields were added, resulting in the version 2 (v2) format.

The Internet Privacy Enhanced Mail (PEM) RFCs, published in 1993, include specifications for a public key infrastructure based on X.509 v1 certificates [RFC 1422]. The experience gained in attempts to deploy RFC 1422 made it clear that the v1 and v2 certificate formats are deficient in several respects. Most importantly, more fields were needed to carry information which PEM design and implementation experience had proven necessary. In response to these new requirements, ISO/IEC, ITU-T and ANSI X9 developed the X.509 version 3 (v3) certificate format. The v3 format extends the v2 format by adding provision for additional extension fields. Particular extension field types may be specified in standards or may be defined and registered by any organization or community. In June 1996, standardization of the basic v3 format was completed [X.509].

ISO/IEC, ITU-T, and ANSI X9 have also developed standard extensions for use in the v3 extensions field [X.509][X9.55]. These extensions can convey such data as additional subject identification information, key attribute information, policy information, and certification path constraints.

However, the ISO/IEC, ITU-T, and ANSI X9 standard extensions are very broad in their applicability. In order to develop interoperable implementations of X.509 v3 systems for Internet use, it is necessary to specify a profile for use of the X.509 v3 extensions tailored for the Internet. It is one goal of this document to specify a profile for Internet WWW, electronic mail, and IPsec applications. Environments with additional requirements may build on this profile or may replace it.

3.2 Certification Paths and Trust

A user of a security service requiring knowledge of a public key generally needs to obtain and validate a certificate containing the required public key. If the public key user does not already hold an assured copy of the public key of the CA that signed the certificate, the CA’s name, and related information (such as the validity period or name constraints), then it might need an additional certificate to obtain that public key. In general, a chain of multiple certificates may be needed, comprising a certificate of the public key owner (the end entity) signed by one CA, and zero or more additional certificates of CAs signed by other CAs. Such chains, called certification paths, are required because a public key user is only initialized with a limited number of assured CA public keys.

There are different ways in which CAs might be configured in order for public key users to be able to find certification paths. For PEM, RFC 1422 defined a rigid hierarchical structure of CAs. There are three types of PEM certification authority:

(a) Internet Policy Registration Authority (IPRA): This authority, operated under the auspices of the Internet Society, acts as the root of the PEM certification hierarchy at level 1. It issues certificates only for the next level of authorities, PCAs. All certification paths start with the IPRA.

(b) Policy Certification Authorities (PCAs): PCAs are at level 2 of the hierarchy, each PCA being certified by the IPRA. A PCA shall establish and publish a statement of its policy with respect to certifying users or subordinate certification authorities. Distinct PCAs aim to satisfy different user needs. For example, one PCA (an organizational PCA) might support the general electronic mail needs of commercial organizations, and another PCA (a high-assurance PCA) might have a more stringent policy designed for satisfying legally binding digital signature requirements.

(c) Certification Authorities (CAs): CAs are at level 3 of the hierarchy and can also be at lower levels. Those at level 3 are certified by PCAs. CAs represent, for example, particular organizations, particular organizational units (e.g., departments, groups, sections), or particular geographical areas.

RFC 1422 furthermore has a name subordination rule which requires that a CA can only issue certificates for entities whose names are subordinate (in the X.500 naming tree) to the name of the CA itself. The trust associated with a PEM certification path is implied by the PCA name. The name subordination rule ensures that CAs below the PCA are sensibly constrained as to the set of subordinate entities they can certify (e.g., a CA for an organization can only certify entities in that organization’s name tree). Certificate user systems are able to mechanically check that the name subordination rule has been followed.

The RFC 1422 uses the X.509 v1 certificate formats. The limitations of X.509 v1 required imposition of several structural restrictions to clearly associate policy information or restrict the utility of certificates. These restrictions included:

(a) a pure top-down hierarchy, with all certification paths starting from IPRA;

(b) a naming subordination rule restricting the names of a CA's subjects; and

(c) use of the PCA concept, which requires knowledge of individual PCAs to be built into certificate chain verification logic. Knowledge of individual PCAs was required to determine if a chain could be accepted.

With X.509 v3, most of the requirements addressed by RFC 1422 can be addressed using certificate extensions, without a need to restrict the CA structures used. In particular, the certificate extensions relating to certificate policies obviate the need for PCAs and the constraint extensions obviate the need for the name subordination rule. As a result, this document supports a more flexible architecture, including:

(a) Certification paths start with a public key of a CA in a user's own domain, or with the public key of the top of a hierarchy. Starting with the public key of a CA in a user's own domain has certain advantages. In some environments, the local domain is the most trusted.

(b) Name constraints may be imposed through explicit inclusion of a name constraints extension in a certificate, but are not required.

(c) Policy extensions and policy mappings replace the PCA concept, which permits a greater degree of automation. The application can determine if the certification path is acceptable based on the contents of the certificates instead of a priori knowledge of PCAs. This permits automation of certification path processing.

3.3 Revocation

When a certificate is issued, it is expected to be in use for its entire validity period. However, various circumstances may cause a certificate to become invalid prior to the expiration of the validity period. Such circumstances include change of name, change of association between subject and CA (e.g., an employee terminates employment with an organization), and compromise or suspected compromise of the corresponding private key. Under such circumstances, the CA needs to revoke the certificate.

X.509 defines one method of certificate revocation. This method involves each CA periodically issuing a signed data structure called a certificate revocation list (CRL). A CRL is a time stamped list identifying revoked certificates which is signed by a CA or CRL issuer and made freely available in a public repository. Each revoked certificate is identified in a CRL by its certificate serial number. When a certificate-using system uses a certificate (e.g., for verifying a remote user's digital signature), that system not only checks the certificate signature and validity but also acquires a suitably-recent CRL and checks that the certificate serial number is not on that CRL. The meaning of "suitably-recent" may vary with local policy, but it usually means the most recently-issued CRL. A new CRL is issued on a regular periodic basis (e.g., hourly, daily, or weekly). An entry is added to the CRL as part of the next update following notification of revocation. An entry MUST NOT be removed from the CRL until it appears on one regularly scheduled CRL issued beyond the revoked certificate's validity period.

An advantage of this revocation method is that CRLs may be distributed by exactly the same means as certificates themselves, namely, via untrusted servers and untrusted communications.

One limitation of the CRL revocation method, using untrusted communications and servers, is that the time granularity of revocation is limited to the CRL issue period. For example, if a revocation is reported now, that revocation will not be reliably notified to certificate-using systems until all currently issued CRLs are updated -- this may be up to one hour, one day, or one week depending on the frequency that CRLs are issued.

As with the X.509 v3 certificate format, in order to facilitate interoperable implementations from multiple vendors, the X.509 v2 CRL format needs to be profiled for Internet use. It is one goal of this document to specify that profile. However, this profile does not require the issuance of CRLs. Message formats and protocols supporting on-line revocation notification are defined in other PKIX specifications. On-line methods of revocation notification may be applicable in some environments as an alternative to the X.509 CRL. On-line revocation checking may significantly reduce the latency between a revocation report and the distribution of the information to relying parties. Once the CA accepts a revocation report as authentic and valid, any query to the on-line service will correctly reflect the certificate validation impacts of the revocation. However, these methods impose new security requirements: the certificate validator needs to trust the on-line validation service while the repository does not need to be trusted.

3.4 Operational Protocols

Operational protocols are required to deliver certificates and CRLs (or status information) to certificate using client systems. Provisions are needed for a variety of different means of certificate and CRL delivery, including distribution procedures based on LDAP, HTTP, FTP, and X.500. Operational protocols supporting these functions are defined in other PKIX specifications. These specifications may include definitions of message formats and procedures for supporting all of the above operational environments, including definitions of or references to appropriate MIME content types.

3.5 Management Protocols

Management protocols are required to support on-line interactions between PKI user and management entities. For example, a management protocol might be used between a CA and a client system with which a key pair is associated, or between two CAs which cross-certify each other. The set of functions which potentially need to be supported by management protocols include:

(a) registration: This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user.

(b) initialization: Before a client system can operate securely it is necessary to install key materials which have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.

Furthermore, a client typically needs to be initialized with its own key pair(s).

(c) certification: This is the process in which a CA issues a certificate for a user’s public key, and returns that certificate to the user’s client system and/or posts that certificate in a repository.

(d) key pair recovery: As an option, user client key materials (e.g., a user’s private key used for encryption purposes) may be backed up by a CA or a key backup system. If a user needs to recover these backed up key materials (e.g., as a result of a forgotten password or a lost key chain file), an on-line protocol exchange may be needed to support such recovery.

(e) key pair update: All key pairs need to be updated regularly, i.e., replaced with a new key pair, and new certificates issued.

(f) revocation request: An authorized person advises a CA of an abnormal situation requiring certificate revocation.

(g) cross-certification: Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA which contains a CA signature key used for issuing certificates.

Note that on-line protocols are not the only way of implementing the above functions. For all functions there are off-line methods of achieving the same result, and this specification does not mandate use of on-line protocols. For example, when hardware tokens are used, many of the functions may be achieved as part of the physical token delivery. Furthermore, some of the above functions may be combined into one protocol exchange. In particular, two or more of the registration, initialization, and certification functions can be combined into one protocol exchange.

The PKIX series of specifications defines a set of standard message formats supporting the above functions. The protocols for conveying these messages in different environments (e.g., e-mail, file transfer, and WWW) are described in those specifications.

4 Certificate and Certificate Extensions Profile

This section presents a profile for public key certificates that will foster interoperability and a reusable PKI. This section is based upon the X.509 v3 certificate format and the standard certificate extensions defined in [X.509]. The ISO/IEC and ITU-T documents use the 1997 version of ASN.1; while this document uses the 1988 ASN.1 syntax, the encoded certificate and standard extensions are equivalent. This section also defines private extensions required to support a PKI for the Internet community.

Certificates may be used in a wide range of applications and environments covering a broad spectrum of interoperability goals and a broader spectrum of operational and assurance requirements. The goal of this document is to establish a common baseline for generic applications requiring broad interoperability and limited special purpose requirements. In particular, the emphasis will be on supporting the use of X.509 v3 certificates for informal Internet electronic mail, IPsec, and WWW applications.

4.1 Basic Certificate Fields

The X.509 v3 certificate basic syntax is as follows. For signature calculation, the data that is to be signed is encoded using the ASN.1 distinguished encoding rules (DER) [X.690]. ASN.1 DER encoding is a tag, length, value encoding system for each element.

```
   Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }

   TBSCertificate  ::=  SEQUENCE  {
        version         [0]  EXPLICIT Version DEFAULT v1,
        serialNumber         CertificateSerialNumber,
        signature            AlgorithmIdentifier,
        issuer               Name,
        validity             Validity,
        subject              Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version MUST be v2 or v3
        subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version MUST be v2 or v3
        extensions      [3]  EXPLICIT Extensions OPTIONAL
                             -- If present, version MUST be v3
        }

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

   CertificateSerialNumber  ::=  INTEGER

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

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

   UniqueIdentifier  ::=  BIT STRING

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

   Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
   Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING  }
```

The following items describe the X.509 v3 certificate for use in the Internet.

4.1.1 Certificate Fields

The Certificate is a SEQUENCE of three required fields. The fields are described in detail in the following subsections.

4.1.1.1 tbsCertificate

The field contains the names of the subject and issuer, a public key associated with the subject, a validity period, and other associated information. The fields are described in detail in section 4.1.2; the tbsCertificate usually includes extensions which are described in section 4.2.

4.1.1.2 signatureAlgorithm

The signatureAlgorithm field contains the identifier for the cryptographic algorithm used by the CA to sign this certificate. [PKIXALGS] lists supported signature algorithms, but other signature algorithms MAY also be supported.

An algorithm identifier is defined by the following ASN.1 structure:

```
   AlgorithmIdentifier  ::=  SEQUENCE  {
        algorithm               OBJECT IDENTIFIER,
        parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

The algorithm identifier is used to identify a cryptographic algorithm. The OBJECT IDENTIFIER component identifies the algorithm (such as DSA with SHA-1). The contents of the optional parameters field will vary according to the algorithm identified.

This field MUST contain the same algorithm identifier as the signature field in the sequence tbsCertificate (section 4.1.2.3).

4.1.1.3 signatureValue

The signatureValue field contains a digital signature computed upon the ASN.1 DER encoded tbsCertificate. The ASN.1 DER encoded tbsCertificate is used as the input to the signature function. This signature value is encoded as a BIT STRING and included in the signature field. The details of this process are specified for each of algorithms listed in [PKIXALGS].

By generating this signature, a CA certifies the validity of the information in the tbsCertificate field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

4.1.2 TBSCertificate

The sequence TBSCertificate contains information associated with the subject of the certificate and the CA who issued it. Every TBSCertificate contains the names of the subject and issuer, a public key associated with the subject, a validity period, a version number, and a serial number; some MAY contain optional unique identifier fields. The remainder of this section describes the syntax and semantics of these fields. A TBSCertificate usually includes extensions. Extensions for the Internet PKI are described in Section 4.2.

4.1.2.1 Version

This field describes the version of the encoded certificate. When extensions are used, as expected in this profile, version MUST be 3 (value is 2). If no extensions are present, but a UniqueIdentifier is present, the version SHOULD be 2 (value is 1); however version MAY be 3. If only basic fields are present, the version SHOULD be 1 (the value is omitted from the certificate as the default value); however the version MAY be 2 or 3.

Implementations SHOULD be prepared to accept any version certificate. At a minimum, conforming implementations MUST recognize version 3 certificates.

Generation of version 2 certificates is not expected by implementations based on this profile.

4.1.2.2 Serial number

The serial number MUST be a positive integer assigned by the CA to each certificate. It MUST be unique for each certificate issued by a given CA (i.e., the issuer name and serial number identify a unique certificate). CAs MUST force the serialNumber to be a non-negative integer.

Given the uniqueness requirements above, serial numbers can be expected to contain long integers. Certificate users MUST be able to handle serialNumber values up to 20 octets. Conformant CAs MUST NOT use serialNumber values longer than 20 octets.

Note: Non-conforming CAs may issue certificates with serial numbers that are negative, or zero. Certificate users SHOULD be prepared to gracefully handle such certificates.

4.1.2.3 Signature

This field contains the algorithm identifier for the algorithm used by the CA to sign the certificate.

This field MUST contain the same algorithm identifier as the signatureAlgorithm field in the sequence Certificate (section 4.1.1.2). The contents of the optional parameters field will vary according to the algorithm identified. [PKIXALGS] lists the supported signature algorithms, but other signature algorithms MAY also be supported.

4.1.2.4 Issuer

The issuer field identifies the entity who has signed and issued the certificate. The issuer field MUST contain a non-empty distinguished name (DN). The issuer field is defined as the X.501 type Name [X.501]. Name is defined by the following ASN.1 structures:

```
Name ::= CHOICE {
  RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::=
  SET OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
  type     AttributeType,
  value    AttributeValue }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY DEFINED BY AttributeType
```

```
   DirectoryString ::= CHOICE {
         teletexString           TeletexString (SIZE (1..MAX)),
         printableString         PrintableString (SIZE (1..MAX)),
         universalString         UniversalString (SIZE (1..MAX)),
         utf8String              UTF8String (SIZE (1..MAX)),
         bmpString               BMPString (SIZE (1..MAX)) }
```

The Name describes a hierarchical name composed of attributes, such as country name, and corresponding values, such as US. The type of the component AttributeValue is determined by the AttributeType; in general it will be a DirectoryString.

The DirectoryString type is defined as a choice of PrintableString, TeletexString, BMPString, UTF8String, and UniversalString. The UTF8String encoding [RFC 2279] is the preferred encoding, and all certificates issued after December 31, 2003 MUST use the UTF8String encoding of DirectoryString (except as noted below). Until that date, conforming CAs MUST choose from the following options when creating a distinguished name, including their own:

(a) if the character set is sufficient, the string MAY be represented as a PrintableString;

(b) failing (a), if the BMPString character set is sufficient the string MAY be represented as a BMPString; and

(c) failing (a) and (b), the string MUST be represented as a UTF8String. If (a) or (b) is satisfied, the CA MAY still choose to represent the string as a UTF8String.

Exceptions to the December 31, 2003 UTF8 encoding requirements are as follows:

(a) CAs MAY issue "name rollover" certificates to support an orderly migration to UTF8String encoding. Such certificates would include the CA’s UTF8String encoded name as issuer and and the old name encoding as subject, or vice-versa.

(b) As stated in section 4.1.2.6, the subject field MUST be populated with a non-empty distinguished name matching the contents of the issuer field in all certificates issued by the subject CA regardless of encoding.

The TeletexString and UniversalString are included for backward compatibility, and SHOULD NOT be used for certificates for new subjects. However, these types MAY be used in certificates where the name was previously established. Certificate users SHOULD be prepared to receive certificates with these types.

In addition, many legacy implementations support names encoded in the ISO 8859-1 character set (Latin1String) [ISO 8859-1] but tag them as TeletexString. TeletexString encodes a larger character set than ISO 8859-1, but it encodes some characters differently. Implementations SHOULD be prepared to handle both encodings.

As noted above, distinguished names are composed of attributes. This specification does not restrict the set of attribute types that may appear in names. However, conforming implementations MUST be prepared to receive certificates with issuer names containing the set of attribute types defined below. This specification RECOMMENDS support for additional attribute types.

Standard sets of attributes have been defined in the X.500 series of specifications [X.520]. Implementations of this specification MUST be prepared to receive the following standard attribute types in issuer and subject (section 4.1.2.6) names:

* country,
* organization,
* organizational-unit,
* distinguished name qualifier,
* state or province name,
* common name (e.g., "Susan Housley"), and
* serial number.

In addition, implementations of this specification SHOULD be prepared to receive the following standard attribute types in issuer and subject names:

* locality,
* title,
* surname,
* given name,
* initials,
* pseudonym, and
* generation qualifier (e.g., "Jr.", "3rd", or "IV").

The syntax and associated object identifiers (OIDs) for these attribute types are provided in the ASN.1 modules in Appendix A.

In addition, implementations of this specification MUST be prepared to receive the domainComponent attribute, as defined in [RFC 2247]. The Domain Name System (DNS) provides a hierarchical resource labeling system. This attribute provides a convenient mechanism for organizations that wish to use DNs that parallel their DNS names. This is not a replacement for the dNSName component of the alternative name field. Implementations are not required to convert such names into DNS names. The syntax and associated OID for this attribute type is provided in the ASN.1 modules in Appendix A.

Certificate users MUST be prepared to process the issuer distinguished name and subject distinguished name (section 4.1.2.6) fields to perform name chaining for certification path validation (section 6). Name chaining is performed by matching the issuer distinguished name in one certificate with the subject name in a CA certificate.

This specification requires only a subset of the name comparison functionality specified in the X.500 series of specifications. Conforming implementations are REQUIRED to implement the following name comparison rules:

(a) attribute values encoded in different types (e.g., PrintableString and BMPString) MAY be assumed to represent different strings;

(b) attribute values in types other than PrintableString are case sensitive (this permits matching of attribute values as binary objects);

(c) attribute values in PrintableString are not case sensitive (e.g., "Marianne Swanson" is the same as "MARIANNE SWANSON"); and

(d) attribute values in PrintableString are compared after removing leading and trailing white space and converting internal substrings of one or more consecutive white space characters to a single space.

These name comparison rules permit a certificate user to validate certificates issued using languages or encodings unfamiliar to the certificate user.

In addition, implementations of this specification MAY use these comparison rules to process unfamiliar attribute types for name chaining. This allows implementations to process certificates with unfamiliar attributes in the issuer name.

Note that the comparison rules defined in the X.500 series of specifications indicate that the character sets used to encode data in distinguished names are irrelevant. The characters themselves are compared without regard to encoding. Implementations of this profile are permitted to use the comparison algorithm defined in the X.500 series. Such an implementation will recognize a superset of name matches recognized by the algorithm specified above.

4.1.2.5 Validity

The certificate validity period is the time interval during which the CA warrants that it will maintain information about the status of the certificate. The field is represented as a SEQUENCE of two dates: the date on which the certificate validity period begins (notBefore) and the date on which the certificate validity period ends (notAfter). Both notBefore and notAfter may be encoded as UTCTime or GeneralizedTime.

CAs conforming to this profile MUST always encode certificate validity dates through the year 2049 as UTCTime; certificate validity dates in 2050 or later MUST be encoded as GeneralizedTime.

The validity period for a certificate is the period of time from notBefore through notAfter, inclusive.

4.1.2.5.1 UTCTime

The universal time type, UTCTime, is a standard ASN.1 type intended for representation of dates and time. UTCTime specifies the year through the two low order digits and time is specified to the precision of one minute or one second. UTCTime includes either Z (for Zulu, or Greenwich Mean Time) or a time differential.

For the purposes of this profile, UTCTime values MUST be expressed Greenwich Mean Time (Zulu) and MUST include seconds (i.e., times are YYMMDDHHMMSSZ), even where the number of seconds is zero. Conforming systems MUST interpret the year field (YY) as follows:

> Where YY is greater than or equal to 50, the year SHALL be interpreted as 19YY; and
>
> Where YY is less than 50, the year SHALL be interpreted as 20YY.

4.1.2.5.2 GeneralizedTime

The generalized time type, GeneralizedTime, is a standard ASN.1 type for variable precision representation of time. Optionally, the GeneralizedTime field can include a representation of the time differential between local and Greenwich Mean Time.

For the purposes of this profile, GeneralizedTime values MUST be expressed Greenwich Mean Time (Zulu) and MUST include seconds (i.e., times are YYYYMMDDHHMMSSZ), even where the number of seconds is zero. GeneralizedTime values MUST NOT include fractional seconds.

4.1.2.6 Subject

The subject field identifies the entity associated with the public key stored in the subject public key field. The subject name MAY be carried in the subject field and/or the subjectAltName extension. If the subject is a CA (e.g., the basic constraints extension, as discussed in 4.2.1.10, is present and the value of cA is TRUE), then the subject field MUST be populated with a non-empty distinguished name matching the contents of the issuer field (section 4.1.2.4) in all certificates issued by the subject CA. If the subject is a CRL issuer (e.g., the key usage extension, as discussed in 4.2.1.3, is present and the value of cRLSign is TRUE) then the subject field MUST be populated with a non-empty distinguished name matching the contents of the issuer field (section 4.1.2.4) in all CRLs issued by the subject CRL issuer. If subject naming information is present only in the subjectAltName extension (e.g., a key bound only to an email address or URI), then the subject name MUST be an empty sequence and the subjectAltName extension MUST be critical.

Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN). The DN MUST be unique for each subject entity certified by the one CA as defined by the issuer name field. A CA MAY issue more than one certificate with the same DN to the same subject entity.

The subject name field is defined as the X.501 type Name. Implementation requirements for this field are those defined for the issuer field (section 4.1.2.4). When encoding attribute values of type DirectoryString, the encoding rules for the issuer field MUST be implemented. Implementations of this specification MUST be prepared to receive subject names containing the attribute types required for the issuer field. Implementations of this specification SHOULD be prepared to receive subject names containing the recommended attribute types for the issuer field. The syntax and associated object identifiers (OIDs) for these attribute types are provided in the ASN.1 modules in Appendix A. Implementations of this specification MAY use these comparison rules to process unfamiliar attribute types (i.e., for name chaining). This allows implementations to process certificates with unfamiliar attributes in the subject name.

In addition, legacy implementations exist where an RFC 822 name is embedded in the subject distinguished name as an EmailAddress attribute. The attribute value for EmailAddress is of type IA5String to permit inclusion of the character '@', which is not part of the PrintableString character set. EmailAddress attribute values are not case sensitive (e.g., "fanfeedback@redsox.com" is the same as "FANFEEDBACK@REDSOX.COM").

Conforming implementations generating new certificates with electronic mail addresses MUST use the rfc822Name in the subject alternative name field (section 4.2.1.7) to describe such identities. Simultaneous inclusion of the EmailAddress attribute in the subject distinguished name to support legacy implementations is deprecated but permitted.

4.1.2.7 Subject Public Key Info

This field is used to carry the public key and identify the algorithm with which the key is used (e.g., RSA, DSA, or Diffie-Hellman). The algorithm is identified using the AlgorithmIdentifier structure specified in section 4.1.1.2. The object identifiers for the supported algorithms and the methods for encoding the public key materials (public key and parameters) are specified in [PKIXALGS].

4.1.2.8 Unique Identifiers

These fields MUST only appear if the version is 2 or 3 (section 4.1.2.1). These fields MUST NOT appear if the version is 1. The subject and issuer unique identifiers are present in the certificate to handle the possibility of reuse of subject and/or issuer names over time. This profile RECOMMENDS that names not be reused for different entities and that Internet certificates not make use of unique identifiers. CAs conforming to this profile SHOULD NOT generate certificates with unique identifiers. Applications conforming to this profile SHOULD be capable of parsing unique identifiers.

4.1.2.9 Extensions

This field MUST only appear if the version is 3 (section 4.1.2.1). If present, this field is a SEQUENCE of one or more certificate extensions. The format and content of certificate extensions in the Internet PKI is defined in section 4.2.

4.2 Certificate Extensions

The extensions defined for X.509 v3 certificates provide methods for associating additional attributes with users or public keys and for managing a certification hierarchy. The X.509 v3 certificate format also allows communities to define private extensions to carry information unique to those communities. Each extension in a certificate is designated as either critical or non-critical. A certificate using system MUST reject the certificate if it encounters a critical extension it does not recognize; however, a non-critical extension MAY be ignored if it is not recognized. The following sections present recommended extensions used within Internet certificates and standard locations for information. Communities may elect to use additional extensions; however, caution ought to be exercised in adopting any critical extensions in certificates which might prevent use in a general context.

Each extension includes an OID and an ASN.1 structure. When an extension appears in a certificate, the OID appears as the field extnID and the corresponding ASN.1 encoded structure is the value of the octet string extnValue. A certificate MUST NOT include more than one instance of a particular extension. For example, a certificate may contain only one authority key identifier extension (section 4.2.1.1). An extension includes the boolean critical, with a default value of FALSE. The text for each extension specifies the acceptable values for the critical field.

Conforming CAs MUST support key identifiers (sections 4.2.1.1 and 4.2.1.2), basic constraints (section 4.2.1.10), key usage (section 4.2.1.3), and certificate policies (section 4.2.1.5) extensions. If the CA issues certificates with an empty sequence for the subject field, the CA MUST support the subject alternative name extension (section 4.2.1.7). Support for the remaining extensions is OPTIONAL. Conforming CAs MAY support extensions that are not identified within this specification; certificate issuers are cautioned that marking such extensions as critical may inhibit interoperability.

At a minimum, applications conforming to this profile MUST recognize the following extensions: key usage (section 4.2.1.3), certificate policies (section 4.2.1.5), the subject alternative name (section 4.2.1.7), basic constraints (section 4.2.1.10), name constraints (section 4.2.1.11), policy constraints (section 4.2.1.12), extended key usage (section 4.2.1.13), and inhibit any-policy (section 4.2.1.15).

In addition, applications conforming to this profile SHOULD recognize the authority and subject key identifier (sections 4.2.1.1 and 4.2.1.2), and policy mapping (section 4.2.1.6) extensions.

4.2.1 Standard Extensions

This section identifies standard certificate extensions defined in [X.509] for use in the Internet PKI. Each extension is associated with an OID defined in [X.509]. These OIDs are members of the id-ce arc, which is defined by the following:

```
id-ce   OBJECT IDENTIFIER ::=  { joint-iso-ccitt(2) ds(5) 29 }
```

4.2.1.1 Authority Key Identifier

The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a certificate. This extension is used where an issuer has multiple signing keys (either due to multiple concurrent key pairs or due to changeover). The identification MAY be based on either the key identifier (the subject key identifier in the issuer’s certificate) or on the issuer name and serial number.

The keyIdentifier field of the authorityKeyIdentifier extension MUST be included in all certificates generated by conforming CAs to facilitate certification path construction. There is one exception; where a CA distributes its public key in the form of a "self-signed" certificate, the authority key identifier MAY be omitted. The signature on a self-signed certificate is generated with the private key associated with the certificate’s subject public key. (This proves that the issuer possesses both the public and private keys.) In this case, the subject and authority key identifiers would be identical, but only the subject key identifier is needed for certification path building.

The value of the keyIdentifier field SHOULD be derived from the public key used to verify the certificate’s signature or a method that generates unique values. Two common methods for generating key identifiers from the public key, and one common method for generating unique values, are described in section 4.2.1.2. Where a key identifier has not been previously established, this specification RECOMMENDS use of one of these methods for generating keyIdentifiers. Where a key identifier has been previously established, the CA SHOULD use the previously established identifier.

This profile RECOMMENDS support for the key identifier method by all certificate users.

This extension MUST NOT be marked critical.

```
id-ce-authorityKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 35 }

AuthorityKeyIdentifier ::= SEQUENCE {
   keyIdentifier             [0] KeyIdentifier           OPTIONAL,
   authorityCertIssuer       [1] GeneralNames            OPTIONAL,
   authorityCertSerialNumber [2] CertificateSerialNumber OPTIONAL  }

KeyIdentifier ::= OCTET STRING
```

4.2.1.2 Subject Key Identifier

The subject key identifier extension provides a means of identifying certificates that contain a particular public key.

To facilitate certification path construction, this extension MUST appear in all conforming CA certificates, that is, all certificates including the basic constraints extension (section 4.2.1.10) where the value of cA is TRUE. The value of the subject key identifier MUST be the value placed in the key identifier field of the Authority Key Identifier extension (section 4.2.1.1) of certificates issued by the subject of this certificate.

For CA certificates, subject key identifiers SHOULD be derived from the public key or a method that generates unique values. Two common methods for generating key identifiers from the public key are:

(1) The keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits).

(2) The keyIdentifier is composed of a four bit type field with the value 0100 followed by the least significant 60 bits of the SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bit string bits).

One common method for generating unique values is a monotonically increasing sequence of integers.

For end entity certificates, the subject key identifier extension provides a means for identifying certificates containing the particular public key used in an application. Where an end entity has obtained multiple certificates, especially from multiple CAs, the subject key identifier provides a means to quickly identify the set of certificates containing a particular public key. To assist applications in identifying the appropriate end entity certificate, this extension SHOULD be included in all end entity certificates.

For end entity certificates, subject key identifiers SHOULD be derived from the public key. Two common methods for generating key identifiers from the public key are identified above.

Where a key identifier has not been previously established, this specification RECOMMENDS use of one of these methods for generating keyIdentifiers. Where a key identifier has been previously established, the CA SHOULD use the previously established identifier.

This extension MUST NOT be marked critical.

```
   id-ce-subjectKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 14 }

   SubjectKeyIdentifier ::= KeyIdentifier
```

4.2.1.3 Key Usage

The key usage extension defines the purpose (e.g., encipherment, signature, certificate signing) of the key contained in the certificate. The usage restriction might be employed when a key that could be used for more than one operation is to be restricted. For example, when an RSA key should be used only to verify signatures on objects other than public key certificates and CRLs, the digitalSignature and/or nonRepudiation bits would be asserted. Likewise, when an RSA key should be used only for key management, the keyEncipherment bit would be asserted.

This extension MUST appear in certificates that contain public keys that are used to validate digital signatures on other public key certificates or CRLs. When this extension appears, it SHOULD be marked critical.

```
      id-ce-keyUsage OBJECT IDENTIFIER ::=  { id-ce 15 }

      KeyUsage ::= BIT STRING {
           digitalSignature        (0),
           nonRepudiation          (1),
           keyEncipherment         (2),
           dataEncipherment        (3),
           keyAgreement            (4),
           keyCertSign             (5),
           cRLSign                 (6),
           encipherOnly            (7),
           decipherOnly            (8) }
```

Bits in the KeyUsage type are used as follows:

The digitalSignature bit is asserted when the subject public key is used with a digital signature mechanism to support security services other than certificate signing (bit 5), or CRL signing (bit 6). Digital signature mechanisms are often used for entity authentication and data origin authentication with integrity.

The nonRepudiation bit is asserted when the subject public key is used to verify digital signatures used to provide a non-repudiation service which protects against the signing entity falsely denying some action, excluding certificate or CRL signing. In the case of later conflict, a reliable third party may determine the authenticity of the signed data.

Further distinctions between the digitalSignature and nonRepudiation bits may be provided in specific certificate policies.

The keyEncipherment bit is asserted when the subject public key is used for key transport. For example, when an RSA key is to be used for key management, then this bit is set.

The dataEncipherment bit is asserted when the subject public key is used for enciphering user data, other than cryptographic keys.

The keyAgreement bit is asserted when the subject public key is used for key agreement. For example, when a Diffie-Hellman key is to be used for key management, then this bit is set.

The keyCertSign bit is asserted when the subject public key is used for verifying a signature on public key certificates. If the keyCertSign bit is asserted, then the cA bit in the basic constraints extension (section 4.2.1.10) MUST also be asserted.

The cRLSign bit is asserted when the subject public key is used for verifying a signature on certificate revocation list (e.g., a CRL, delta CRL, or an ARL). This bit MUST be asserted in certificates that are used to verify signatures on CRLs.

The meaning of the encipherOnly bit is undefined in the absence of the keyAgreement bit. When the encipherOnly bit is asserted and the keyAgreement bit is also set, the subject public key may be used only for enciphering data while performing key agreement.

The meaning of the decipherOnly bit is undefined in the absence of the keyAgreement bit. When the decipherOnly bit is asserted and the keyAgreement bit is also set, the subject public key may be used only for deciphering data while performing key agreement.

This profile does not restrict the combinations of bits that may be set in an instantiation of the keyUsage extension. However, appropriate values for keyUsage extensions for particular algorithms are specified in [PKIXALGS].

4.2.1.4 Private Key Usage Period

This extension SHOULD NOT be used within the Internet PKI. CAs conforming to this profile MUST NOT generate certificates that include a critical private key usage period extension.

The private key usage period extension allows the certificate issuer to specify a different validity period for the private key than the certificate. This extension is intended for use with digital signature keys. This extension consists of two optional components, notBefore and notAfter. The private key associated with the certificate SHOULD NOT be used to sign objects before or after the times specified by the two components, respectively. CAs conforming to this profile MUST NOT generate certificates with private key usage period extensions unless at least one of the two components is present and the extension is non-critical.

Where used, notBefore and notAfter are represented as GeneralizedTime and MUST be specified and interpreted as defined in section 4.1.2.5.2.

```
id-ce-privateKeyUsagePeriod OBJECT IDENTIFIER ::=  { id-ce 16 }

PrivateKeyUsagePeriod ::= SEQUENCE {
     notBefore       [0]     GeneralizedTime OPTIONAL,
     notAfter        [1]     GeneralizedTime OPTIONAL }
```

4.2.1.5 Certificate Policies

The certificate policies extension contains a sequence of one or more policy information terms, each of which consists of an object identifier (OID) and optional qualifiers. Optional qualifiers, which MAY be present, are not expected to change the definition of the policy.

In an end entity certificate, these policy information terms indicate the policy under which the certificate has been issued and the purposes for which the certificate may be used. In a CA certificate, these policy information terms limit the set of policies for certification paths which include this certificate. When a CA does not wish to limit the set of policies for certification paths which include this certificate, it MAY assert the special policy anyPolicy, with a value of { 2 5 29 32 0 }.

Applications with specific policy requirements are expected to have a list of those policies which they will accept and to compare the policy OIDs in the certificate to that list. If this extension is critical, the path validation software MUST be able to interpret this extension (including the optional qualifier), or MUST reject the certificate.

To promote interoperability, this profile RECOMMENDS that policy information terms consist of only an OID. Where an OID alone is insufficient, this profile strongly recommends that use of qualifiers be limited to those identified in this section. When qualifiers are used with the special policy anyPolicy, they MUST be limited to the qualifiers identified in this section.

This specification defines two policy qualifier types for use by certificate policy writers and certificate issuers. The qualifier types are the CPS Pointer and User Notice qualifiers.

The CPS Pointer qualifier contains a pointer to a Certification Practice Statement (CPS) published by the CA. The pointer is in the form of a URI. Processing requirements for this qualifier are a local matter. No action is mandated by this specification regardless of the criticality value asserted for the extension.

User notice is intended for display to a relying party when a certificate is used. The application software SHOULD display all user notices in all certificates of the certification path used, except that if a notice is duplicated only one copy need be displayed. To prevent such duplication, this qualifier SHOULD only be present in end entity certificates and CA certificates issued to other organizations.

The user notice has two optional fields: the noticeRef field and the explicitText field.

> The noticeRef field, if used, names an organization and identifies, by number, a particular textual statement prepared by that organization. For example, it might identify the organization "CertsRUs" and notice number 1. In a typical implementation, the application software will have a notice file containing the current set of notices for CertsRUs; the application will extract the notice text from the file and display it. Messages MAY be multilingual, allowing the software to select the particular language message for its own environment.
>
> An explicitText field includes the textual statement directly in the certificate. The explicitText field is a string with a maximum size of 200 characters.

If both the noticeRef and explicitText options are included in the one qualifier and if the application software can locate the notice text indicated by the noticeRef option, then that text SHOULD be displayed; otherwise, the explicitText string SHOULD be displayed.

Note: While the explicitText has a maximum size of 200 characters, some non-conforming CAs exceed this limit. Therefore, certificate users SHOULD gracefully handle explicitText with more than 200 characters.

```
   id-ce-certificatePolicies OBJECT IDENTIFIER ::=  { id-ce 32 }

   anyPolicy OBJECT IDENTIFIER ::= { id-ce-certificate-policies 0 }

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

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

   CertPolicyId ::= OBJECT IDENTIFIER

   PolicyQualifierInfo ::= SEQUENCE {
        policyQualifierId  PolicyQualifierId,
        qualifier          ANY DEFINED BY policyQualifierId }

   -- policyQualifierIds for Internet policy qualifiers

   id-qt          OBJECT IDENTIFIER ::=  { id-pkix 2 }
   id-qt-cps      OBJECT IDENTIFIER ::=  { id-qt 1 }
   id-qt-unotice  OBJECT IDENTIFIER ::=  { id-qt 2 }

   PolicyQualifierId ::=
        OBJECT IDENTIFIER ( id-qt-cps | id-qt-unotice )

   Qualifier ::= CHOICE {
        cPSuri           CPSuri,
        userNotice       UserNotice }

   CPSuri ::= IA5String

   UserNotice ::= SEQUENCE {
        noticeRef        NoticeReference OPTIONAL,
        explicitText     DisplayText OPTIONAL}

   NoticeReference ::= SEQUENCE {
        organization     DisplayText,
        noticeNumbers    SEQUENCE OF INTEGER }

   DisplayText ::= CHOICE {
        ia5String        IA5String      (SIZE (1..200)),
        visibleString    VisibleString  (SIZE (1..200)),
        bmpString        BMPString      (SIZE (1..200)),
        utf8String       UTF8String     (SIZE (1..200)) }
```

4.2.1.6 Policy Mappings

This extension is used in CA certificates. It lists one or more pairs of OIDs; each pair includes an issuerDomainPolicy and a subjectDomainPolicy. The pairing indicates the issuing CA considers its issuerDomainPolicy equivalent to the subject CA's subjectDomainPolicy.

The issuing CA's users might accept an issuerDomainPolicy for certain applications. The policy mapping defines the list of policies associated with the subject CA that may be accepted as comparable to the issuerDomainPolicy.

Each issuerDomainPolicy named in the policy mapping extension SHOULD also be asserted in a certificate policies extension in the same certificate. Policies SHOULD NOT be mapped either to or from the special value anyPolicy (section 4.2.1.5).

This extension MAY be supported by CAs and/or applications, and it MUST be non-critical.

```
id-ce-policyMappings OBJECT IDENTIFIER ::=  { id-ce 33 }

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

4.2.1.7 Subject Alternative Name

The subject alternative names extension allows additional identities to be bound to the subject of the certificate. Defined options include an Internet electronic mail address, a DNS name, an IP address, and a uniform resource identifier (URI). Other options exist, including completely local definitions. Multiple name forms, and multiple instances of each name form, MAY be included. Whenever such identities are to be bound into a certificate, the subject alternative name (or issuer alternative name) extension MUST be used; however, a DNS name MAY be represented in the subject field using the domainComponent attribute as described in section 4.1.2.4.

Because the subject alternative name is considered to be definitively bound to the public key, all parts of the subject alternative name MUST be verified by the CA.

Further, if the only subject identity included in the certificate is an alternative name form (e.g., an electronic mail address), then the subject distinguished name MUST be empty (an empty sequence), and the subjectAltName extension MUST be present. If the subject field contains an empty sequence, the subjectAltName extension MUST be marked critical.

When the subjectAltName extension contains an Internet mail address, the address MUST be included as an rfc822Name. The format of an rfc822Name is an "addr-spec" as defined in RFC 822 [RFC 822]. An addr-spec has the form "local-part@domain". Note that an addr-spec has no phrase (such as a common name) before it, has no comment (text surrounded in parentheses) after it, and is not surrounded by "<" and ">". Note that while upper and lower case letters are allowed in an RFC 822 addr-spec, no significance is attached to the case.

When the subjectAltName extension contains a iPAddress, the address MUST be stored in the octet string in "network byte order," as specified in RFC 791 [RFC 791]. The least significant bit (LSB) of each octet is the LSB of the corresponding byte in the network address. For IP Version 4, as specified in RFC 791, the octet string MUST contain exactly four octets. For IP Version 6, as specified in RFC 1883, the octet string MUST contain exactly sixteen octets [RFC 1883].

When the subjectAltName extension contains a domain name system label, the domain name MUST be stored in the dNSName (an IA5String). The name MUST be in the "preferred name syntax," as specified by RFC 1034 [RFC 1034]. Note that while upper and lower case letters are allowed in domain names, no signifigance is attached to the case. In addition, while the string " " is a legal domain name, subjectAltName extensions with a dNSName of " " MUST NOT be used. Finally, the use of the DNS representation for Internet mail addresses (wpolk.nist.gov instead of wpolk@nist.gov) MUST NOT be used; such identities are to be encoded as rfc822Name.

Note: work is currently underway to specify domain names in international character sets. Such names will likely not be accommodated by IA5String. Once this work is complete, this profile will be revisited and the appropriate functionality will be added.

When the subjectAltName extension contains a URI, the name MUST be stored in the uniformResourceIdentifier (an IA5String). The name MUST NOT be a relative URL, and it MUST follow the URL syntax and encoding rules specified in [RFC 1738]. The name MUST include both a scheme (e.g., "http" or "ftp") and a scheme-specific-part. The scheme-specific-part MUST include a fully qualified domain name or IP address as the host.

As specified in [RFC 1738], the scheme name is not case-sensitive (e.g., "http" is equivalent to "HTTP"). The host part is also not case-sensitive, but other components of the scheme-specific-part may be case-sensitive. When comparing URIs, conforming implementations MUST compare the scheme and host without regard to case, but assume the remainder of the scheme-specific-part is case sensitive.

When the subjectAltName extension contains a DN in the directoryName, the DN MUST be unique for each subject entity certified by the one CA as defined by the issuer name field. A CA MAY issue more than one certificate with the same DN to the same subject entity.

The subjectAltName MAY carry additional name types through the use of the otherName field. The format and semantics of the name are indicated through the OBJECT IDENTIFIER in the type-id field. The name itself is conveyed as value field in otherName. For example, Kerberos [RFC 1510] format names can be encoded into the otherName, using using a Kerberos 5 principal name OID and a SEQUENCE of the Realm and the PrincipalName.

Subject alternative names MAY be constrained in the same manner as subject distinguished names using the name constraints extension as described in section 4.2.1.11.

If the subjectAltName extension is present, the sequence MUST contain at least one entry. Unlike the subject field, conforming CAs MUST NOT issue certificates with subjectAltNames containing empty GeneralName fields. For example, an rfc822Name is represented as an IA5String. While an empty string is a valid IA5String, such an rfc822Name is not permitted by this profile. The behavior of clients that encounter such a certificate when processing a certificication path is not defined by this profile.

Finally, the semantics of subject alternative names that include wildcard characters (e.g., as a placeholder for a set of names) are not addressed by this specification. Applications with specific requirements MAY use such names, but they must define the semantics.

```
   id-ce-subjectAltName OBJECT IDENTIFIER ::=  { id-ce 17 }

   SubjectAltName ::= GeneralNames

   GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName
   GeneralName ::= CHOICE {
        otherName                       [0]     OtherName,
        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 }

   OtherName ::= SEQUENCE {
        type-id    OBJECT IDENTIFIER,
        value      [0] EXPLICIT ANY DEFINED BY type-id }

   EDIPartyName ::= SEQUENCE {
        nameAssigner            [0]     DirectoryString OPTIONAL,
        partyName               [1]     DirectoryString }
```

4.2.1.8 Issuer Alternative Names

As with 4.2.1.7, this extension is used to associate Internet style identities with the certificate issuer. Issuer alternative names MUST be encoded as in 4.2.1.7.

Where present, this extension SHOULD NOT be marked critical.

```
   id-ce-issuerAltName OBJECT IDENTIFIER ::=  { id-ce 18 }

   IssuerAltName ::= GeneralNames
```

4.2.1.9 Subject Directory Attributes

The subject directory attributes extension is used to convey identification attributes (e.g., nationality) of the subject. The extension is defined as a sequence of one or more attributes. This extension MUST be non-critical.

```
   id-ce-subjectDirectoryAttributes OBJECT IDENTIFIER ::=  { id-ce 9 }

   SubjectDirectoryAttributes ::= SEQUENCE SIZE (1..MAX) OF Attribute
```

4.2.1.10 Basic Constraints

The basic constraints extension identifies whether the subject of the certificate is a CA and the maximum depth of valid certification paths that include this certificate.

The cA boolean indicates whether the certified public key belongs to a CA. If the cA boolean is not asserted, then the keyCertSign bit in the key usage extension MUST NOT be asserted.

The pathLenConstraint field is meaningful only if the cA boolean is asserted and the key usage extension asserts the keyCertSign bit (section 4.2.1.3). In this case, it gives the maximum number of non-self-issued intermediate certificates that may follow this certificate in a valid certification path. A certificate is self-issued if the DNs that appear in the subject and issuer fields are identical and are not empty. (Note: The last certificate in the certification path is not an intermediate certificate, and is not included in this limit. Usually, the last certificate is an end entity certificate, but it can be a CA certificate.) A pathLenConstraint of zero indicates that only one more certificate may follow in a valid certification path. Where it appears, the pathLenConstraint field MUST be greater than or equal to zero. Where pathLenConstraint does not appear, no limit is imposed.

This extension MUST appear as a critical extension in all CA certificates that contain public keys used to validate digital signatures on certificates. This extension MAY appear as a critical or non-critical extension in CA certificates that contain public keys used exclusively for purposes other than validating digital signatures on certificates. Such CA certificates include ones that contain public keys used exclusively for validating digital signatures on CRLs and ones that contain key management public keys used with certificate enrollment protocols. This extension MAY appear as a critical or non-critical extension in end entity certificates.

CAs MUST NOT include the pathLenConstraint field unless the cA boolean is asserted and the key usage extension asserts the keyCertSign bit.

```
   id-ce-basicConstraints OBJECT IDENTIFIER ::=  { id-ce 19 }

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

4.2.1.11 Name Constraints

The name constraints extension, which MUST be used only in a CA certificate, indicates a name space within which all subject names in subsequent certificates in a certification path MUST be located. Restrictions apply to the subject distinguished name and apply to subject alternative names. Restrictions apply only when the specified name form is present. If no name of the type is in the certificate, the certificate is acceptable.

Name constraints are not applied to certificates whose issuer and subject are identical (unless the certificate is the final certificate in the path). (This could prevent CAs that use name constraints from employing self-issued certificates to implement key rollover.)

Restrictions are defined in terms of permitted or excluded name subtrees. Any name matching a restriction in the excludedSubtrees field is invalid regardless of information appearing in the permittedSubtrees. This extension MUST be critical.

Within this profile, the minimum and maximum fields are not used with any name forms, thus minimum MUST be zero, and maximum MUST be absent.

For URIs, the constraint applies to the host part of the name. The constraint MAY specify a host or a domain. Examples would be "foo.bar.com"; and ".xyz.com". When the the constraint begins with a period, it MAY be expanded with one or more subdomains. That is, the constraint ".xyz.com" is satisfied by both abc.xyz.com and abc.def.xyz.com. However, the constraint ".xyz.com" is not satisfied by "xyz.com". When the constraint does not begin with a period, it specifies a host.

A name constraint for Internet mail addresses MAY specify a particular mailbox, all addresses at a particular host, or all mailboxes in a domain. To indicate a particular mailbox, the constraint is the complete mail address. For example, "root@xyz.com" indicates the root mailbox on the host "xyz.com". To indicate all Internet mail addresses on a particular host, the constraint is specified as the host name. For example, the constraint "xyz.com" is satisfied by any mail address at the host "xyz.com". To specify any address within a domain, the constraint is specified with a leading period (as with URIs). For example, ".xyz.com" indicates all the Internet mail addresses in the domain "xyz.com", but not Internet mail addresses on the host "xyz.com".

DNS name restrictions are expressed as foo.bar.com. Any DNS name that can be constructed by simply adding to the left hand side of the name satisfies the name constraint. For example, www.foo.bar.com would satisfy the constraint but foo1.bar.com would not.

Legacy implementations exist where an RFC 822 name is embedded in the subject distinguished name in an attribute of type EmailAddress (section 4.1.2.6). When rfc822 names are constrained, but the certificate does not include a subject alternative name, the rfc822 name constraint MUST be applied to the attribute of type EmailAddress in the subject distinguished name. The ASN.1 syntax for EmailAddress and the corresponding OID are supplied in Appendix A.

Restrictions of the form directoryName MUST be applied to the subject field in the certificate and to the subjectAltName extensions of type directoryName. Restrictions of the form x400Address MUST be applied to subjectAltName extensions of type x400Address.

When applying restrictions of the form directoryName, an implementation MUST compare DN attributes. At a minimum, implementations MUST perform the DN comparison rules specified in Section 4.1.2.4. CAs issuing certificates with a restriction of the form directoryName SHOULD NOT rely on implementation of the full ISO DN name comparison algorithm. This implies name restrictions MUST be stated identically to the encoding used in the subject field or subjectAltName extension.

The syntax of iPAddress MUST be as described in section 4.2.1.7 with the following additions specifically for Name Constraints. For IPv4 addresses, the ipAddress field of generalName MUST contain eight (8) octets, encoded in the style of RFC 1519 (CIDR) to represent an address range [RFC 1519]. For IPv6 addresses, the ipAddress field MUST contain 32 octets similarly encoded. For example, a name constraint for "class C" subnet 10.9.8.0 is represented as the octets 0A 09 08 00 FF FF FF 00, representing the CIDR notation 10.9.8.0/255.255.255.0.

The syntax and semantics for name constraints for otherName, ediPartyName, and registeredID are not defined by this specification.

```
   id-ce-nameConstraints OBJECT IDENTIFIER ::=  { id-ce 30 }

   NameConstraints ::= 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)
```

4.2.1.12 Policy Constraints

The policy constraints extension can be used in certificates issued to CAs. The policy constraints extension constrains path validation in two ways. It can be used to prohibit policy mapping or require that each certificate in a path contain an acceptable policy identifier.

If the inhibitPolicyMapping field is present, the value indicates the number of additional certificates that may appear in the path before policy mapping is no longer permitted. For example, a value of one indicates that policy mapping may be processed in certificates issued by the subject of this certificate, but not in additional certificates in the path.

If the requireExplicitPolicy field is present, the value of requireExplicitPolicy indicates the number of additional certificates that may appear in the path before an explicit policy is required for the entire path. When an explicit policy is required, it is necessary for all certificates in the path to contain an acceptable policy identifier in the certificate policies extension. An acceptable policy identifier is the identifier of a policy required by the user of the certification path or the identifier of a policy which has been declared equivalent through policy mapping.

Conforming CAs MUST NOT issue certificates where policy constraints is a empty sequence. That is, at least one of the inhibitPolicyMapping field or the requireExplicitPolicy field MUST be present. The behavior of clients that encounter a empty policy constraints field is not addressed in this profile.

This extension MAY be critical or non-critical.

```
id-ce-policyConstraints OBJECT IDENTIFIER ::=  { id-ce 36 }

PolicyConstraints ::= SEQUENCE {
     requireExplicitPolicy           [0] SkipCerts OPTIONAL,
     inhibitPolicyMapping            [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)
```

4.2.1.13 Extended Key Usage

This extension indicates one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the key usage extension. In general, this extension will appear only in end entity certificates. This extension is defined as follows:

```
   id-ce-extKeyUsage OBJECT IDENTIFIER ::= { id-ce 37 }

   ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId

   KeyPurposeId ::= OBJECT IDENTIFIER
```

Key purposes may be defined by any organization with a need. Object identifiers used to identify key purposes MUST be assigned in accordance with IANA or ITU-T Recommendation X.660 [X.660].

This extension MAY, at the option of the certificate issuer, be either critical or non-critical.

If the extension is present, then the certificate MUST only be used for one of the purposes indicated. If multiple purposes are indicated the application need not recognize all purposes indicated, as long as the intended purpose is present. Certificate using applications MAY require that a particular purpose be indicated in order for the certificate to be acceptable to that application.

If a CA includes extended key usages to satisfy such applications, but does not wish to restrict usages of the key, the CA can include the special keyPurposeID anyExtendedKeyUsage. If the anyExtendedKeyUsage keyPurposeID is present, the extension SHOULD NOT be critical.

If a certificate contains both a key usage extension and an extended key usage extension, then both extensions MUST be processed independently and the certificate MUST only be used for a purpose consistent with both extensions. If there is no purpose consistent with both extensions, then the certificate MUST NOT be used for any purpose.

The following key usage purposes are defined:

```
   anyExtendedKeyUsage OBJECT IDENTIFIER ::= { id-ce-extKeyUsage 0 }

   id-kp OBJECT IDENTIFIER ::= { id-pkix 3 }

   id-kp-serverAuth             OBJECT IDENTIFIER ::= { id-kp 1 }
   -- TLS WWW server authentication
   -- Key usage bits that may be consistent: digitalSignature,
   -- keyEncipherment or keyAgreement

   id-kp-clientAuth             OBJECT IDENTIFIER ::= { id-kp 2 }
   -- TLS WWW client authentication
   -- Key usage bits that may be consistent: digitalSignature
   -- and/or keyAgreement
   id-kp-codeSigning             OBJECT IDENTIFIER ::= { id-kp 3 }
   -- Signing of downloadable executable code
   -- Key usage bits that may be consistent: digitalSignature

   id-kp-emailProtection         OBJECT IDENTIFIER ::= { id-kp 4 }
   -- E-mail protection
   -- Key usage bits that may be consistent: digitalSignature,
   -- nonRepudiation, and/or (keyEncipherment or keyAgreement)

   id-kp-timeStamping            OBJECT IDENTIFIER ::= { id-kp 8 }
   -- Binding the hash of an object to a time
   -- Key usage bits that may be consistent: digitalSignature
   -- and/or nonRepudiation

   id-kp-OCSPSigning            OBJECT IDENTIFIER ::= { id-kp 9 }
   -- Signing OCSP responses
   -- Key usage bits that may be consistent: digitalSignature
   -- and/or nonRepudiation
```

4.2.1.14 CRL Distribution Points

The CRL distribution points extension identifies how CRL information is obtained. The extension SHOULD be non-critical, but this profile RECOMMENDS support for this extension by CAs and applications. Further discussion of CRL management is contained in section 5.

The cRLDistributionPoints extension is a SEQUENCE of DistributionPoint. A DistributionPoint consists of three fields, each of which is optional: distributionPoint, reasons, and cRLIssuer. While each of these fields is optional, a DistributionPoint MUST NOT consist of only the reasons field; either distributionPoint or cRLIssuer MUST be present. If the certificate issuer is not the CRL issuer, then the cRLIssuer field MUST be present and contain the Name of the CRL issuer. If the certificate issuer is also the CRL issuer, then the cRLIssuer field MUST be omitted and the distributionPoint field MUST be present. If the distributionPoint field is omitted, cRLIssuer MUST be present and include a Name corresponding to an X.500 or LDAP directory entry where the CRL is located.

When the distributionPoint field is present, it contains either a SEQUENCE of general names or a single value, nameRelativeToCRLIssuer. If the cRLDistributionPoints extension contains a general name of type URI, the following semantics MUST be assumed: the URI is a pointer to the current CRL for the associated reasons and will be issued by the associated cRLIssuer. The expected values for the URI are those defined in 4.2.1.7. Processing rules for other values are not defined by this specification.

If the DistributionPointName contains multiple values, each name describes a different mechanism to obtain the same CRL. For example, the same CRL could be available for retrieval through both LDAP and HTTP.

If the DistributionPointName contains the single value nameRelativeToCRLIssuer, the value provides a distinguished name fragment. The fragment is appended to the X.500 distinguished name of the CRL issuer to obtain the distribution point name. If the cRLIssuer field in the DistributionPoint is present, then the name fragment is appended to the distinguished name that it contains; otherwise, the name fragment is appended to the certificate issuer distinguished name. The DistributionPointName MUST NOT use the nameRealtiveToCRLIssuer alternative when cRLIssuer contains more than one distinguished name.

If the DistributionPoint omits the reasons field, the CRL MUST include revocation information for all reasons.

The cRLIssuer identifies the entity who signs and issues the CRL. If present, the cRLIssuer MUST contain at least one an X.500 distinguished name (DN), and MAY also contain other name forms. Since the cRLIssuer is compared to the CRL issuer name, the X.501 type Name MUST follow the encoding rules for the issuer name field in the certificate (section 4.1.2.4).

```
id-ce-cRLDistributionPoints OBJECT IDENTIFIER ::=  { id-ce 31 }

CRLDistributionPoints ::= 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),
        privilegeWithdrawn      (7),
        aACompromise            (8) }
```

4.2.1.15 Inhibit Any-Policy

The inhibit any-policy extension can be used in certificates issued to CAs. The inhibit any-policy indicates that the special anyPolicy OID, with the value { 2 5 29 32 0 }, is not considered an explicit match for other certificate policies. The value indicates the number of additional certificates that may appear in the path before anyPolicy is no longer permitted. For example, a value of one indicates that anyPolicy may be processed in certificates issued by the subject of this certificate, but not in additional certificates in the path.

This extension MUST be critical.

```
   id-ce-inhibitAnyPolicy OBJECT IDENTIFIER ::=  { id-ce 54 }

   InhibitAnyPolicy ::= SkipCerts

   SkipCerts ::= INTEGER (0..MAX)
```

4.2.1.16 Freshest CRL (a.k.a. Delta CRL Distribution Point)

The freshest CRL extension identifies how delta CRL information is obtained. The extension MUST be non-critical. Further discussion of CRL management is contained in section 5.

The same syntax is used for this extension and the cRLDistributionPoints extension, and is described in section 4.2.1.14. The same conventions apply to both extensions.

```
   id-ce-freshestCRL OBJECT IDENTIFIER ::=  { id-ce 46 }

   FreshestCRL ::= CRLDistributionPoints
```

4.2.2 Private Internet Extensions

This section defines two extensions for use in the Internet Public Key Infrastructure. These extensions may be used to direct applications to on-line information about the issuing CA or the subject. As the information may be available in multiple forms, each extension is a sequence of IA5String values, each of which represents a URI. The URI implicitly specifies the location and format of the information and the method for obtaining the information.

An object identifier is defined for the private extension. The object identifier associated with the private extension is defined under the arc id-pe within the arc id-pkix. Any future extensions defined for the Internet PKI are also expected to be defined under the arc id-pe.

```
   id-pkix  OBJECT IDENTIFIER  ::=
            { iso(1) identified-organization(3) dod(6) internet(1)
                    security(5) mechanisms(5) pkix(7) }

   id-pe  OBJECT IDENTIFIER  ::=  { id-pkix 1 }
```

4.2.2.1 Authority Information Access

The authority information access extension indicates how to access CA information and services for the issuer of the certificate in which the extension appears. Information and services may include on-line validation services and CA policy data. (The location of CRLs is not specified in this extension; that information is provided by the cRLDistributionPoints extension.) This extension may be included in end entity or CA certificates, and it MUST be non-critical.

```
id-pe-authorityInfoAccess OBJECT IDENTIFIER ::= { id-pe 1 }

AuthorityInfoAccessSyntax  ::=
        SEQUENCE SIZE (1..MAX) OF AccessDescription

AccessDescription  ::=  SEQUENCE {
        accessMethod          OBJECT IDENTIFIER,
        accessLocation        GeneralName  }

id-ad OBJECT IDENTIFIER ::= { id-pkix 48 }

id-ad-caIssuers OBJECT IDENTIFIER ::= { id-ad 2 }

id-ad-ocsp OBJECT IDENTIFIER ::= { id-ad 1 }
```

Each entry in the sequence AuthorityInfoAccessSyntax describes the format and location of additional information provided by the CA that issued the certificate in which this extension appears. The type and format of the information is specified by the accessMethod field; the accessLocation field specifies the location of the information. The retrieval mechanism may be implied by the accessMethod or specified by accessLocation.

This profile defines two accessMethod OIDs: id-ad-caIssuers and id-ad-ocsp.

The id-ad-caIssuers OID is used when the additional information lists CAs that have issued certificates superior to the CA that issued the certificate containing this extension. The referenced CA issuers description is intended to aid certificate users in the selection of a certification path that terminates at a point trusted by the certificate user.

When id-ad-caIssuers appears as accessMethod, the accessLocation field describes the referenced description server and the access protocol to obtain the referenced description. The accessLocation field is defined as a GeneralName, which can take several forms. Where the information is available via http, ftp, or ldap, accessLocation MUST be a uniformResourceIdentifier. Where the information is available via the Directory Access Protocol (DAP), accessLocation MUST be a directoryName. The entry for that directoryName contains CA certificates in the crossCertificatePair attribute. When the information is available via electronic mail, accessLocation MUST be an rfc822Name. The semantics of other id-ad-caIssuers accessLocation name forms are not defined.

The id-ad-ocsp OID is used when revocation information for the certificate containing this extension is available using the Online Certificate Status Protocol (OCSP) [RFC 2560].

When id-ad-ocsp appears as accessMethod, the accessLocation field is the location of the OCSP responder, using the conventions defined in [RFC 2560].

Additional access descriptors may be defined in other PKIX specifications.

4.2.2.2 Subject Information Access

The subject information access extension indicates how to access information and services for the subject of the certificate in which the extension appears. When the subject is a CA, information and services may include certificate validation services and CA policy data. When the subject is an end entity, the information describes the type of services offered and how to access them. In this case, the contents of this extension are defined in the protocol specifications for the suported services. This extension may be included in subject or CA certificates, and it MUST be non-critical.

```
   id-pe-subjectInfoAccess OBJECT IDENTIFIER ::= { id-pe 11 }

   SubjectInfoAccessSyntax  ::=
           SEQUENCE SIZE (1..MAX) OF AccessDescription

   AccessDescription  ::=  SEQUENCE {
           accessMethod          OBJECT IDENTIFIER,
           accessLocation        GeneralName  }
```

Each entry in the sequence SubjectInfoAccessSyntax describes the format and location of additional information provided by the subject of the certificate in which this extension appears. The type and format of the information is specified by the accessMethod field; the accessLocation field specifies the location of the information. The retrieval mechanism may be implied by the accessMethod or specified by accessLocation.

This profile defines one access method to be used when the subject is a CA, and one access method to be used when the subject is an end entity. Additional access methods may be defined in the future in the protocol specifications for other services.

The id-ad-caRepository OID is used when the subject is a CA, and publishes its certificates and CRLs (if issued) in a repository. The accessLocation field is defined as a GeneralName, which can take several forms. Where the information is available via http, ftp, or ldap, accessLocation MUST be a uniformResourceIdentifier. Where the information is available via the directory access protocol (dap), accessLocation MUST be a directoryName. When the information is available via electronic mail, accessLocation MUST be an rfc822Name. The semantics of other name forms of of accessLocation (when accessMethod is id-ad-caRepository) are not defined by this specification.

The id-ad-timeStamping OID is used when the subject offers timestamping services using the Time Stamp Protocol defined in [PKIXTSA]. Where the timestamping services are available via http or ftp, accessLocation MUST be a uniformResourceIdentifier. Where the timestamping services are available via electronic mail, accessLocation MUST be an rfc822Name. Where timestamping services are available using TCP/IP, the dNSName or ipAddress name forms may be used. The semantics of other name forms of accessLocation (when accessMethod is id-ad-timeStamping) are not defined by this specification.

Additional access descriptors may be defined in other PKIX specifications.

```
id-ad OBJECT IDENTIFIER ::= { id-pkix 48 }

id-ad-caRepository OBJECT IDENTIFIER ::= { id-ad 5 }

id-ad-timeStamping OBJECT IDENTIFIER ::= { id-ad 3 }
```

5 CRL and CRL Extensions Profile

As discussed above, one goal of this X.509 v2 CRL profile is to foster the creation of an interoperable and reusable Internet PKI. To achieve this goal, guidelines for the use of extensions are specified, and some assumptions are made about the nature of information included in the CRL.

CRLs may be used in a wide range of applications and environments covering a broad spectrum of interoperability goals and an even broader spectrum of operational and assurance requirements. This profile establishes a common baseline for generic applications requiring broad interoperability. The profile defines a set of information that can be expected in every CRL. Also, the profile defines common locations within the CRL for frequently used attributes as well as common representations for these attributes.

CRL issuers issue CRLs. In general, the CRL issuer is the CA. CAs publish CRLs to provide status information about the certificates they issued. However, a CA may delegate this responsibility to another trusted authority. Whenever the CRL issuer is not the CA that issued the certificates, the CRL is referred to as an indirect CRL.

Each CRL has a particular scope. The CRL scope is the set of certificates that could appear on a given CRL. For example, the scope could be "all certificates issued by CA X", "all CA certificates issued by CA X", "all certificates issued by CA X that have been revoked for reasons of key compromise and CA compromise", or could be a set of certificates based on arbitrary local information, such as "all certificates issued to the NIST employees located in Boulder".

A complete CRL lists all unexpired certificates, within its scope, that have been revoked for one of the revocation reasons covered by the CRL scope. The CRL issuer MAY also generate delta CRLs. A delta CRL only lists those certificates, within its scope, whose revocation status has changed since the issuance of a referenced complete CRL. The referenced complete CRL is referred to as a base CRL. The scope of a delta CRL MUST be the same as the base CRL that it references.

This profile does not define any private Internet CRL extensions or CRL entry extensions.

Environments with additional or special purpose requirements may build on this profile or may replace it.

Conforming CAs are not required to issue CRLs if other revocation or certificate status mechanisms are provided. When CRLs are issued, the CRLs MUST be version 2 CRLs, include the date by which the next CRL will be issued in the nextUpdate field (section 5.1.2.5), include the CRL number extension (section 5.2.3), and include the authority key identifier extension (section 5.2.1). Conforming applications that support CRLs are REQUIRED to process both version 1 and version 2 complete CRLs that provide revocation information for all certificates issued by one CA. Conforming applications are NOT REQUIRED to support processing of delta CRLs, indirect CRLs, or CRLs with a scope other than all certificates issued by one CA.

5.1 CRL Fields

The X.509 v2 CRL syntax is as follows. 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.

```
CertificateList  ::=  SEQUENCE  {
     tbsCertList          TBSCertList,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }
```

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

   -- Version, Time, CertificateSerialNumber, and Extensions
   -- are all defined in the ASN.1 in section 4.1

   -- AlgorithmIdentifier is defined in section 4.1.1.2
```

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

5.1.1 CertificateList Fields

The CertificateList is a SEQUENCE of three required fields. The fields are described in detail in the following subsections.

5.1.1.1 tbsCertList

The first field in the sequence is the tbsCertList. This field is itself a sequence containing the name of the issuer, issue date, issue date of the next list, the optional list of revoked certificates, and optional CRL extensions. When there are no revoked certificates, the revoked certificates list is absent. When one or more certificates are revoked, each entry on the revoked certificate list is defined by a sequence of user certificate serial number, revocation date, and optional CRL entry extensions.

5.1.1.2 signatureAlgorithm

The signatureAlgorithm field contains the algorithm identifier for the algorithm used by the CRL issuer to sign the CertificateList. The field is of type AlgorithmIdentifier, which is defined in section 4.1.1.2. [PKIXALGS] lists the supported algorithms for this specification, but other signature algorithms MAY also be supported.

This field MUST contain the same algorithm identifier as the signature field in the sequence tbsCertList (section 5.1.2.2).

5.1.1.3 signatureValue

The signatureValue field contains a digital signature computed upon the ASN.1 DER encoded tbsCertList. The ASN.1 DER encoded tbsCertList is used as the input to the signature function. This signature value is encoded as a BIT STRING and included in the CRL signatureValue field. The details of this process are specified for each of the supported algorithms in [PKIXALGS].

CAs that are also CRL issuers MAY use one private key to digitally sign certificates and CRLs, or MAY use separate private keys to digitally sign certificates and CRLs. When separate private keys are employed, each of the public keys associated with these private keys is placed in a separate certificate, one with the keyCertSign bit set in the key usage extension, and one with the cRLSign bit set in the key usage extension (section 4.2.1.3). When separate private keys are employed, certificates issued by the CA contain one authority key identifier, and the corresponding CRLs contain a different authority key identifier. The use of separate CA certificates for validation of certificate signatures and CRL signatures can offer improved security characteristics; however, it imposes a burden on applications, and it might limit interoperability. Many applications construct a certification path, and then validate the certification path (section 6). CRL checking in turn requires a separate certification path to be constructed and validated for the CA’s CRL signature validation certificate. Applications that perform CRL checking MUST support certification path validation when certificates and CRLs are digitally signed with the same CA private key. These applications SHOULD support certification path validation when certificates and CRLs are digitally signed with different CA private keys.

5.1.2 Certificate List "To Be Signed"

The certificate list to be signed, or TBSCertList, is a sequence of required and optional fields. The required fields identify the CRL issuer, the algorithm used to sign the CRL, the date and time the CRL was issued, and the date and time by which the CRL issuer will issue the next CRL.

Optional fields include lists of revoked certificates and CRL extensions. The revoked certificate list is optional to support the case where a CA has not revoked any unexpired certificates that it has issued. The profile requires conforming CRL issuers to use the CRL number and authority key identifier CRL extensions in all CRLs issued.

5.1.2.1 Version

This optional field describes the version of the encoded CRL. When extensions are used, as required by this profile, this field MUST be present and MUST specify version 2 (the integer value is 1).

5.1.2.2 Signature

This field contains the algorithm identifier for the algorithm used to sign the CRL. [PKIXALGS] lists OIDs for the most popular signature algorithms used in the Internet PKI.

This field MUST contain the same algorithm identifier as the signatureAlgorithm field in the sequence CertificateList (section 5.1.1.2).

5.1.2.3 Issuer Name

The issuer name identifies the entity who has signed and issued the CRL. The issuer identity is carried in the issuer name field. Alternative name forms may also appear in the issuerAltName extension (section 5.2.2). The issuer name field MUST contain an X.500 distinguished name (DN). The issuer name field is defined as the X.501 type Name, and MUST follow the encoding rules for the issuer name field in the certificate (section 4.1.2.4).

5.1.2.4 This Update

This field indicates the issue date of this CRL. ThisUpdate may be encoded as UTCTime or GeneralizedTime.

CRL issuers conforming to this profile MUST encode thisUpdate as UTCTime for dates through the year 2049. CRL issuers conforming to this profile MUST encode thisUpdate as GeneralizedTime for dates in the year 2050 or later.

Where encoded as UTCTime, thisUpdate MUST be specified and interpreted as defined in section 4.1.2.5.1. Where encoded as GeneralizedTime, thisUpdate MUST be specified and interpreted as defined in section 4.1.2.5.2.

5.1.2.5 Next Update

This 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. CRL issuers SHOULD issue CRLs with a nextUpdate time equal to or later than all previous CRLs. nextUpdate may be encoded as UTCTime or GeneralizedTime.

This profile requires inclusion of nextUpdate in all CRLs issued by conforming CRL issuers. Note that the ASN.1 syntax of TBSCertList describes this field as OPTIONAL, which is consistent with the ASN.1 structure defined in [X.509]. The behavior of clients processing CRLs which omit nextUpdate is not specified by this profile.

CRL issuers conforming to this profile MUST encode nextUpdate as UTCTime for dates through the year 2049. CRL issuers conforming to this profile MUST encode nextUpdate as GeneralizedTime for dates in the year 2050 or later.

Where encoded as UTCTime, nextUpdate MUST be specified and interpreted as defined in section 4.1.2.5.1. Where encoded as GeneralizedTime, nextUpdate MUST be specified and interpreted as defined in section 4.1.2.5.2.

5.1.2.6 Revoked Certificates

When there are no revoked certificates, the revoked certificates list MUST be absent. Otherwise, revoked certificates are listed by their serial numbers. Certificates revoked by the CA are uniquely identified by the certificate serial number. The date on which the revocation occurred is specified. The time for revocationDate MUST be expressed as described in section 5.1.2.4. Additional information may be supplied in CRL entry extensions; CRL entry extensions are discussed in section 5.3.

5.1.2.7 Extensions

This field may only appear if the version is 2 (section 5.1.2.1). If present, this field is a sequence of one or more CRL extensions. CRL extensions are discussed in section 5.2.

5.2 CRL Extensions

The extensions defined by ANSI X9, ISO/IEC, and ITU-T for X.509 v2 CRLs [X.509] [X9.55] provide methods for associating additional attributes with CRLs. The X.509 v2 CRL format also allows communities to define private extensions to carry information unique to those communities. Each extension in a CRL may be designated as critical or non-critical. A CRL validation MUST fail if it encounters a critical extension which it does not know how to process. However, an unrecognized non-critical extension may be ignored. The following subsections present those extensions used within Internet CRLs. Communities may elect to include extensions in CRLs which are not defined in this specification. However, caution should be exercised in adopting any critical extensions in CRLs which might be used in a general context.

Conforming CRL issuers are REQUIRED to include the authority key identifier (section 5.2.1) and the CRL number (section 5.2.3) extensions in all CRLs issued.

5.2.1 Authority Key Identifier

The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a CRL. The identification can be based on either the key identifier (the subject key identifier in the CRL signer’s certificate) or on the issuer name and serial number. This extension is especially useful where an issuer has more than one signing key, either due to multiple concurrent key pairs or due to changeover.

Conforming CRL issuers MUST use the key identifier method, and MUST include this extension in all CRLs issued.

The syntax for this CRL extension is defined in section 4.2.1.1.

5.2.2 Issuer Alternative Name

The issuer alternative names extension allows additional identities to be associated with the issuer of the CRL. Defined options include an rfc822 name (electronic mail address), a DNS name, an IP address, and a URI. Multiple instances of a name and multiple name forms may be included. Whenever such identities are used, the issuer alternative name extension MUST be used; however, a DNS name MAY be represented in the issuer field using the domainComponent attribute as described in section 4.1.2.4.

The issuerAltName extension SHOULD NOT be marked critical.

The OID and syntax for this CRL extension are defined in section 4.2.1.8.

5.2.3 CRL Number

The CRL number is a non-critical CRL extension which conveys a monotonically increasing sequence number for a given CRL scope and CRL issuer. This extension allows users to easily determine when a particular CRL supersedes another CRL. CRL numbers also support the identification of complementary complete CRLs and delta CRLs. CRL issuers conforming to this profile MUST include this extension in all CRLs.

If a CRL issuer generates delta CRLs in addition to complete CRLs for a given scope, the complete CRLs and delta CRLs MUST share one numbering sequence. If a delta CRL and a complete CRL that cover the same scope are issued at the same time, they MUST have the same CRL number and provide the same revocation information. That is, the combination of the delta CRL and an acceptable complete CRL MUST provide the same revocation information as the simultaneously issued complete CRL.

If a CRL issuer generates two CRLs (two complete CRLs, two delta CRLs, or a complete CRL and a delta CRL) for the same scope at different times, the two CRLs MUST NOT have the same CRL number. That is, if the this update field (section 5.1.2.4) in the two CRLs are not identical, the CRL numbers MUST be different.

Given the requirements above, CRL numbers can be expected to contain long integers. CRL verifiers MUST be able to handle CRLNumber values up to 20 octets. Conformant CRL issuers MUST NOT use CRLNumber values longer than 20 octets.

```
id-ce-cRLNumber OBJECT IDENTIFIER ::= { id-ce 20 }

CRLNumber ::= INTEGER (0..MAX)
```

5.2.4 Delta CRL Indicator

The delta CRL indicator is a critical CRL extension that identifies a CRL as being a delta CRL. Delta CRLs contain updates to revocation information previously distributed, rather than all the information that would appear in a complete CRL. The use of delta CRLs can significantly reduce network load and processing time in some environments. Delta CRLs are generally smaller than the CRLs they update, so applications that obtain delta CRLs consume less network bandwidth than applications that obtain the corresponding complete CRLs. Applications which store revocation information in a format other than the CRL structure can add new revocation information to the local database without reprocessing information.

The delta CRL indicator extension contains the single value of type BaseCRLNumber. The CRL number identifies the CRL, complete for a given scope, that was used as the starting point in the generation of this delta CRL. A conforming CRL issuer MUST publish the referenced base CRL as a complete CRL. The delta CRL contains all updates to the revocation status for that same scope. The combination of a delta CRL plus the referenced base CRL is equivalent to a complete CRL, for the applicable scope, at the time of publication of the delta CRL.

When a conforming CRL issuer generates a delta CRL, the delta CRL MUST include a critical delta CRL indicator extension.

When a delta CRL is issued, it MUST cover the same set of reasons and the same set of certificates that were covered by the base CRL it references. That is, the scope of the delta CRL MUST be the same as the scope of the complete CRL referenced as the base. The referenced base CRL and the delta CRL MUST omit the issuing distribution point extension or contain identical issuing distribution point extensions. Further, the CRL issuer MUST use the same private key to sign the delta CRL and any complete CRL that it can be used to update.

An application that supports delta CRLs can construct a CRL that is complete for a given scope by combining a delta CRL for that scope with either an issued CRL that is complete for that scope or a locally constructed CRL that is complete for that scope.

When a delta CRL is combined with a complete CRL or a locally constructed CRL, the resulting locally constructed CRL has the CRL number specified in the CRL number extension found in the delta CRL used in its construction. In addition, the resulting locally constructed CRL has the thisUpdate and nextUpdate times specified in the corresponding fields of the delta CRL used in its construction. In addition, the locally constructed CRL inherits the issuing distribution point from the delta CRL.

A complete CRL and a delta CRL MAY be combined if the following four conditions are satisfied:

(a) The complete CRL and delta CRL have the same issuer.

(b) The complete CRL and delta CRL have the same scope. The two CRLs have the same scope if either of the following conditions are met:

(1) The issuingDistributionPoint extension is omitted from both the complete CRL and the delta CRL.

(2) The issuingDistributionPoint extension is present in both the complete CRL and the delta CRL, and the values for each of the fields in the extensions are the same in both CRLs.

(c) The CRL number of the complete CRL is equal to or greater than the BaseCRLNumber specified in the delta CRL. That is, the complete CRL contains (at a minimum) all the revocation information held by the referenced base CRL.

(d) The CRL number of the complete CRL is less than the CRL number of the delta CRL. That is, the delta CRL follows the complete CRL in the numbering sequence.

CRL issuers MUST ensure that the combination of a delta CRL and any appropriate complete CRL accurately reflects the current revocation status. The CRL issuer MUST include an entry in the delta CRL for each certificate within the scope of the delta CRL whose status has changed since the generation of the referenced base CRL:

(a) If the certificate is revoked for a reason included in the scope of the CRL, list the certificate as revoked.

(b) If the certificate is valid and was listed on the referenced base CRL or any subsequent CRL with reason code certificateHold, and the reason code certificateHold is included in the scope of the CRL, list the certificate with the reason code removeFromCRL.

(c) If the certificate is revoked for a reason outside the scope of the CRL, but the certificate was listed on the referenced base CRL or any subsequent CRL with a reason code included in the scope of this CRL, list the certificate as revoked but omit the reason code.

(d) If the certificate is revoked for a reason outside the scope of the CRL and the certificate was neither listed on the referenced base CRL nor any subsequent CRL with a reason code included in the scope of this CRL, do not list the certificate on this CRL.

The status of a certificate is considered to have changed if it is revoked, placed on hold, released from hold, or if its revocation reason changes.

It is appropriate to list a certificate with reason code removeFromCRL on a delta CRL even if the certificate was not on hold in the referenced base CRL. If the certificate was placed on hold in any CRL issued after the base but before this delta CRL and then released from hold, it MUST be listed on the delta CRL with revocation reason removeFromCRL.

A CRL issuer MAY optionally list a certificate on a delta CRL with reason code removeFromCRL if the notAfter time specified in the certificate precedes the thisUpdate time specified in the delta CRL and the certificate was listed on the referenced base CRL or in any CRL issued after the base but before this delta CRL.

If a certificate revocation notice first appears on a delta CRL, then it is possible for the certificate validity period to expire before the next complete CRL for the same scope is issued. In this case, the revocation notice MUST be included in all subsequent delta CRLs until the revocation notice is included on at least one explicitly issued complete CRL for this scope.

An application that supports delta CRLs MUST be able to construct a current complete CRL by combining a previously issued complete CRL and the most current delta CRL. An application that supports delta CRLs MAY also be able to construct a current complete CRL by combining a previously locally constructed complete CRL and the current delta CRL. A delta CRL is considered to be the current one if the current time is between the times contained in the thisUpdate and nextUpdate fields. Under some circumstances, the CRL issuer may publish one or more delta CRLs before indicated by the nextUpdate field. If more than one current delta CRL for a given scope is encountered, the application SHOULD consider the one with the latest value in thisUpdate to be the most current one.

```
id-ce-deltaCRLIndicator OBJECT IDENTIFIER ::= { id-ce 27 }

BaseCRLNumber ::= CRLNumber
```

5.2.5 Issuing Distribution Point

The issuing distribution point is a critical CRL extension that identifies the CRL distribution point and scope for a particular CRL, and it indicates whether the CRL covers revocation for end entity certificates only, CA certificates only, attribute certificates only,

or a limited set of reason codes. Although the extension is critical, conforming implementations are not required to support this extension.

The CRL is signed using the CRL issuer’s private key. CRL Distribution Points do not have their own key pairs. If the CRL is stored in the X.500 Directory, it is stored in the Directory entry corresponding to the CRL distribution point, which may be different than the Directory entry of the CRL issuer.

The reason codes associated with a distribution point MUST be specified in onlySomeReasons. If onlySomeReasons does not appear, the distribution point MUST contain revocations for all reason codes. CAs may use CRL distribution points to partition the CRL on the basis of compromise and routine revocation. In this case, the revocations with reason code keyCompromise (1), cACompromise (2), and aACompromise (8) appear in one distribution point, and the revocations with other reason codes appear in another distribution point.

If the distributionPoint field is present and contains a URI, the following semantics MUST be assumed: the object is a pointer to the most current CRL issued by this CRL issuer. The URI schemes ftp, http, mailto [RFC1738] and ldap [RFC1778] are defined for this purpose. The URI MUST be an absolute pathname, not a relative pathname, and MUST specify the host.

If the distributionPoint field is absent, the CRL MUST contain entries for all revoked unexpired certificates issued by the CRL issuer, if any, within the scope of the CRL.

The CRL issuer MUST assert the indirectCRL boolean, if the scope of the CRL includes certificates issued by authorities other than the CRL issuer. The authority responsible for each entry is indicated by the certificate issuer CRL entry extension (section 5.3.4).

```
   id-ce-issuingDistributionPoint OBJECT IDENTIFIER ::= { id-ce 28 }

   issuingDistributionPoint ::= SEQUENCE {
        distributionPoint          [0] DistributionPointName OPTIONAL,
        onlyContainsUserCerts      [1] BOOLEAN DEFAULT FALSE,
        onlyContainsCACerts        [2] BOOLEAN DEFAULT FALSE,
        onlySomeReasons            [3] ReasonFlags OPTIONAL,
        indirectCRL                [4] BOOLEAN DEFAULT FALSE,
        onlyContainsAttributeCerts [5] BOOLEAN DEFAULT FALSE }
```

5.2.6 Freshest CRL (a.k.a. Delta CRL Distribution Point)

The freshest CRL extension identifies how delta CRL information for this complete CRL is obtained. The extension MUST be non-critical. This extension MUST NOT appear in delta CRLs.

The same syntax is used for this extension as the cRLDistributionPoints certificate extension, and is described in section 4.2.1.14. However, only the distribution point field is meaningful in this context. The reasons and CRLIssuer fields MUST be omitted from this CRL extension.

Each distribution point name provides the location at which a delta CRL for this complete CRL can be found. The scope of these delta CRLs MUST be the same as the scope of this complete CRL. The contents of this CRL extension are only used to locate delta CRLs; the contents are not used to validate the CRL or the referenced delta CRLs. The encoding conventions defined for distribution points in section 4.2.1.14 apply to this extension.

```
id-ce-freshestCRL OBJECT IDENTIFIER ::=  { id-ce 46 }

FreshestCRL ::= CRLDistributionPoints
```

5.3 CRL Entry Extensions

The CRL entry extensions defined by ISO/IEC, ITU-T, and ANSI X9 for X.509 v2 CRLs provide methods for associating additional attributes with CRL entries [X.509] [X9.55]. The X.509 v2 CRL format also allows communities to define private CRL entry extensions to carry information unique to those communities. Each extension in a CRL entry may be designated as critical or non-critical. A CRL validation MUST fail if it encounters a critical CRL entry extension which it does not know how to process. However, an unrecognized non-critical CRL entry extension may be ignored. The following subsections present recommended extensions used within Internet CRL entries and standard locations for information. Communities may elect to use additional CRL entry extensions; however, caution should be exercised in adopting any critical extensions in CRL entries which might be used in a general context.

All CRL entry extensions used in this specification are non-critical. Support for these extensions is optional for conforming CRL issuers and applications. However, CRL issuers SHOULD include reason codes (section 5.3.1) and invalidity dates (section 5.3.3) whenever this information is available.

5.3.1 Reason Code

The reasonCode is a non-critical CRL entry extension that identifies the reason for the certificate revocation. CRL issuers are strongly encouraged to include meaningful reason codes in CRL entries; however, the reason code CRL entry extension SHOULD be absent instead of using the unspecified (0) reasonCode value.

```
   id-ce-cRLReason OBJECT IDENTIFIER ::= { id-ce 21 }

   -- reasonCode ::= { CRLReason }

   CRLReason ::= ENUMERATED {
        unspecified             (0),
        keyCompromise           (1),
        cACompromise            (2),
        affiliationChanged      (3),
        superseded              (4),
        cessationOfOperation    (5),
        certificateHold         (6),
        removeFromCRL           (8),
        privilegeWithdrawn      (9),
        aACompromise           (10) }
```

5.3.2 Hold Instruction Code

The hold instruction code is a non-critical 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.

```
   id-ce-holdInstructionCode OBJECT IDENTIFIER ::= { id-ce 23 }

   holdInstructionCode ::= OBJECT IDENTIFIER
```

The following instruction codes have been defined. Conforming applications that process this extension MUST recognize the following instruction codes.

```
   holdInstruction    OBJECT IDENTIFIER ::=
                    { iso(1) member-body(2) us(840) x9-57(10040) 2 }

   id-holdinstruction-none   OBJECT IDENTIFIER ::= {holdInstruction 1}
   id-holdinstruction-callissuer
                             OBJECT IDENTIFIER ::= {holdInstruction 2}
   id-holdinstruction-reject OBJECT IDENTIFIER ::= {holdInstruction 3}
```

Conforming applications which encounter an id-holdinstruction-callissuer MUST call the certificate issuer or reject the certificate. Conforming applications which encounter an id-holdinstruction-reject MUST reject the certificate. The hold instruction id-holdinstruction-none is semantically equivalent to the absence of a holdInstructionCode, and its use is strongly deprecated for the Internet PKI.

5.3.3 Invalidity Date

The invalidity date is a non-critical 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, which is the date at which the CA processed the revocation. When a revocation is first posted by a CRL issuer in a CRL, the invalidity date may precede the date of issue of earlier CRLs, but the revocation date SHOULD NOT precede the date of issue of earlier CRLs. Whenever this information is available, CRL issuers are strongly encouraged to share it with CRL users.

The GeneralizedTime values included in this field MUST be expressed in Greenwich Mean Time (Zulu), and MUST be specified and interpreted as defined in section 4.1.2.5.2.

```
id-ce-invalidityDate OBJECT IDENTIFIER ::= { id-ce 24 }

invalidityDate ::=  GeneralizedTime
```

5.3.4 Certificate Issuer

This CRL entry extension identifies the certificate issuer associated with an entry in an indirect CRL, that is, a CRL that has the indirectCRL indicator set in its issuing distribution point extension. If this extension is not present on the first entry in an indirect CRL, the certificate issuer defaults to the CRL issuer. On subsequent entries in an indirect CRL, if this extension is not present, the certificate issuer for the entry is the same as that for the preceding entry. This field is defined as follows:

```
id-ce-certificateIssuer   OBJECT IDENTIFIER ::= { id-ce 29 }

certificateIssuer ::=     GeneralNames
```

If used by conforming CRL issuers, this extension MUST always be critical. If an implementation ignored this extension it could not correctly attribute CRL entries to certificates. This specification RECOMMENDS that implementations recognize this extension.

6 Certification Path Validation

Certification path validation procedures for the Internet PKI are based on the algorithm supplied in [X.509]. Certification path processing verifies the binding between the subject distinguished name and/or subject alternative name and subject public key. The binding is limited by constraints which are specified in the certificates which comprise the path and inputs which are specified by the relying party. The basic constraints and policy constraints extensions allow the certification path processing logic to automate the decision making process.

This section describes an algorithm for validating certification paths. Conforming implementations of this specification are not required to implement this algorithm, but MUST provide functionality equivalent to the external behavior resulting from this procedure. Any algorithm may be used by a particular implementation so long as it derives the correct result.

In section 6.1, the text describes basic path validation. Valid paths begin with certificates issued by a trust anchor. The algorithm requires the public key of the CA, the CA's name, and any constraints upon the set of paths which may be validated using this key.

The selection of a trust anchor is a matter of policy: it could be the top CA in a hierarchical PKI; the CA that issued the verifier's own certificate(s); or any other CA in a network PKI. The path validation procedure is the same regardless of the choice of trust anchor. In addition, different applications may rely on different trust anchor, or may accept paths that begin with any of a set of trust anchor.

Section 6.2 describes methods for using the path validation algorithm in specific implementations. Two specific cases are discussed: the case where paths may begin with one of several trusted CAs; and where compatibility with the PEM architecture is required.

Section 6.3 describes the steps necessary to determine if a certificate is revoked or on hold status when CRLs are the revocation mechanism used by the certificate issuer.

6.1 Basic Path Validation

This text describes an algorithm for X.509 path processing. A conformant implementation MUST include an X.509 path processing procedure that is functionally equivalent to the external behavior of this algorithm. However, support for some of the certificate extensions processed in this algorithm are OPTIONAL for compliant implementations. Clients that do not support these extensions MAY omit the corresponding steps in the path validation algorithm.

For example, clients are NOT REQUIRED to support the policy mapping extension. Clients that do not support this extension MAY omit the path validation steps where policy mappings are processed. Note that clients MUST reject the certificate if it contains an unsupported critical extension.

The algorithm presented in this section validates the certificate with respect to the current date and time. A conformant implementation MAY also support validation with respect to some point in the past. Note that mechanisms are not available for validating a certificate with respect to a time outside the certificate validity period.

The trust anchor is an input to the algorithm. There is no requirement that the same trust anchor be used to validate all certification paths. Different trust anchors MAY be used to validate different paths, as discussed further in Section 6.2.

The primary goal of path validation is to verify the binding between a subject distinguished name or a subject alternative name and subject public key, as represented in the end entity certificate, based on the public key of the trust anchor. This requires obtaining a sequence of certificates that support that binding. The procedure performed to obtain this sequence of certificates is outside the scope of this specification.

To meet this goal, the path validation process verifies, among other things, that a prospective certification path (a sequence of n certificates) satisfies the following conditions:

(a) for all x in {1, ..., n-1}, the subject of certificate x is the issuer of certificate x+1;

(b) certificate 1 is issued by the trust anchor;

(c) certificate n is the certificate to be validated; and

(d) for all x in {1, ..., n}, the certificate was valid at the time in question.

When the trust anchor is provided in the form of a self-signed certificate, this self-signed certificate is not included as part of the prospective certification path. Information about trust anchors are provided as inputs to the certification path validation algorithm (section 6.1.1).

A particular certification path may not, however, be appropriate for all applications. Therefore, an application MAY augment this algorithm to further limit the set of valid paths. The path validation process also determines the set of certificate policies that are valid for this path, based on the certificate policies extension, policy mapping extension, policy constraints extension, and inhibit any-policy extension. To achieve this, the path validation algorithm constructs a valid policy tree. If the set of certificate policies that are valid for this path is not empty, then the result will be a valid policy tree of depth n, otherwise the result will be a null valid policy tree.

A certificate is self-issued if the DNs that appear in the subject and issuer fields are identical and are not empty. In general, the issuer and subject of the certificates that make up a path are different for each certificate. However, a CA may issue a certificate to itself to support key rollover or changes in certificate policies. These self-issued certificates are not counted when evaluating path length or name constraints.

This section presents the algorithm in four basic steps: (1) initialization, (2) basic certificate processing, (3) preparation for the next certificate, and (4) wrap-up. Steps (1) and (4) are performed exactly once. Step (2) is performed for all certificates in the path. Step (3) is performed for all certificates in the path except the final certificate. Figure 2 provides a high-level flowchart of this algorithm.

```
                           +-------+
                           | START |
                           +-------+
                               |
                               V
                       +----------------+
                       | Initialization |
                       +----------------+
                               |
                               +<--------------------+
                               |                     |
                               V                     |
                       +----------------+            |
                       |  Process Cert  |            |
                       +----------------+            |
                               |                     |
                               V                     |
                       +================+            |
                       |  IF Last Cert  |            |
                       |    in Path     |            |
                       +================+            |
                         |            |              |
                    THEN |            | ELSE         |
                         V            V              |
              +----------------+ +----------------+  |
              |    Wrap up     | |  Prepare for   |  |
              +----------------+ |   Next Cert    |  |
                      |          +----------------+  |
                      V               |              |
                  +-------+           +--------------+
                  | STOP  |
                  +-------+
```

Figure 2. Certification Path Processing Flowchart

6.1.1 Inputs

This algorithm assumes the following seven inputs are provided to the path processing logic:

(a) a prospective certification path of length n.

(b) the current date/time.

(c) user-initial-policy-set: A set of certificate policy identifiers naming the policies that are acceptable to the certificate user. The user-initial-policy-set contains the special value any-policy if the user is not concerned about certificate policy.

(d) trust anchor information, describing a CA that serves as a trust anchor for the certification path. The trust anchor information includes:

(1) the trusted issuer name,

(2) the trusted public key algorithm,

(3) the trusted public key, and

(4) optionally, the trusted public key parameters associated with the public key.

The trust anchor information may be provided to the path processing procedure in the form of a self-signed certificate. The trusted anchor information is trusted because it was delivered to the path processing procedure by some trustworthy out-of-band procedure. If the trusted public key algorithm requires parameters, then the parameters are provided along with the trusted public key.

(e) initial-policy-mapping-inhibit, which indicates if policy mapping is allowed in the certification path.

(f) initial-explicit-policy, which indicates if the path must be valid for at least one of the certificate policies in the user-initial-policy-set.

(g) initial-any-policy-inhibit, which indicates whether the anyPolicy OID should be processed if it is included in a certificate.

6.1.2 Initialization

This initialization phase establishes eleven state variables based upon the seven inputs:

(a) valid_policy_tree: A tree of certificate policies with their optional qualifiers; each of the leaves of the tree represents a valid policy at this stage in the certification path validation. If valid policies exist at this stage in the certification path validation, the depth of the tree is equal to the number of certificates in the chain that have been processed. If valid policies do not exist at this stage in the certification path validation, the tree is set to NULL. Once the tree is set to NULL, policy processing ceases.

Each node in the valid_policy_tree includes four data objects: the valid policy, a set of associated policy qualifiers, a set of one or more expected policy values, and a criticality indicator. If the node is at depth x, the components of the node have the following semantics:

(1) The valid_policy is a single policy OID representing a valid policy for the path of length x.

(2) The qualifier_set is a set of policy qualifiers associated with the valid policy in certificate x.

(3) The criticality_indicator indicates whether the certificate policy extension in certificate x was marked as critical.

(4) The expected_policy_set contains one or more policy OIDs that would satisfy this policy in the certificate x+1.

The initial value of the valid_policy_tree is a single node with valid_policy anyPolicy, an empty qualifier_set, an expected_policy_set with the single value anyPolicy, and a criticality_indicator of FALSE. This node is considered to be at depth zero.

Figure 3 is a graphic representation of the initial state of the valid_policy_tree. Additional figures will use this format to describe changes in the valid_policy_tree during path processing.

```
        +----------------+
        |   anyPolicy    |   <---- valid_policy
        +----------------+
        |       {}       |   <---- qualifier_set
        +----------------+
        |     FALSE      |   <---- criticality_indicator
        +----------------+
        |  {anyPolicy}   |   <---- expected_policy_set
        +----------------+
```

Figure 3. Initial value of the valid_policy_tree state variable

(b) permitted_subtrees: A set of root names for each name type (e.g., X.500 distinguished names, email addresses, or ip addresses) defining a set of subtrees within which all subject names in subsequent certificates in the certification path MUST fall. This variable includes a set for each name type: the initial value for the set for Distinguished Names is the set of all Distinguished names; the initial value for the set of RFC822 names is the set of all RFC822 names, etc.

(c) excluded_subtrees: A set of root names for each name type (e.g., X.500 distinguished names, email addresses, or ip addresses) defining a set of subtrees within which no subject name in subsequent certificates in the certification path may fall. This variable includes a set for each name type, and the initial value for each set is empty.

(d) explicit_policy: an integer which indicates if a non-NULL valid_policy_tree is required. The integer indicates the number of non-self-issued certificates to be processed before this requirement is imposed. Once set, this variable may be decreased, but may not be increased. That is, if a certificate in the path requires a non-NULL valid_policy_tree, a later certificate can not remove this requirement. If initial-explicit-policy is set, then the initial value is 0, otherwise the initial value is n+1.

(e) inhibit_any-policy: an integer which indicates whether the anyPolicy policy identifier is considered a match. The integer indicates the number of non-self-issued certificates to be processed before the anyPolicy OID, if asserted in a certificate, is ignored. Once set, this variable may be decreased, but may not be increased. That is, if a certificate in the path inhibits processing of anyPolicy, a later certificate can not permit it. If initial-any-policy-inhibit is set, then the initial value is 0, otherwise the initial value is n+1.

(f) policy_mapping: an integer which indicates if policy mapping is permitted. The integer indicates the number of non-self-issued certificates to be processed before policy mapping is inhibited. Once set, this variable may be decreased, but may not be increased. That is, if a certificate in the path specifies policy mapping is not permitted, it can not be overridden by a later certificate. If initial-policy-mapping-inhibit is set, then the initial value is 0, otherwise the initial value is n+1.

(g) working_public_key_algorithm: the digital signature algorithm used to verify the signature of a certificate. The working_public_key_algorithm is initialized from the trusted public key algorithm provided in the trust anchor information.

(h) working_public_key: the public key used to verify the signature of a certificate. The working_public_key is initialized from the trusted public key provided in the trust anchor information.

(i) working_public_key_parameters: parameters associated with the current public key, that may be required to verify a signature (depending upon the algorithm). The working_public_key_parameters variable is initialized from the trusted public key parameters provided in the trust anchor information.

(j) working_issuer_name: the issuer distinguished name expected in the next certificate in the chain. The working_issuer_name is initialized to the trusted issuer provided in the trust anchor information.

(k) max_path_length: this integer is initialized to n, is decremented for each non-self-issued certificate in the path, and may be reduced to the value in the path length constraint field within the basic constraints extension of a CA certificate.

Upon completion of the initialization steps, perform the basic certificate processing steps specified in 6.1.3.

6.1.3 Basic Certificate Processing

The basic path processing actions to be performed for certificate i (for all i in [1..n]) are listed below.

(a) Verify the basic certificate information. The certificate MUST satisfy each of the following:

(1) The certificate was signed with the working_public_key_algorithm using the working_public_key and the working_public_key_parameters.

(2) The certificate validity period includes the current time.

(3) At the current time, the certificate is not revoked and is not on hold status. This may be determined by obtaining the appropriate CRL (section 6.3), status information, or by out-of-band mechanisms.

(4) The certificate issuer name is the working_issuer_name.

(b) If certificate i is self-issued and it is not the final certificate in the path, skip this step for certificate i. Otherwise, verify that the subject name is within one of the permitted_subtrees for X.500 distinguished names, and verify that each of the alternative names in the subjectAltName extension (critical or non-critical) is within one of the permitted_subtrees for that name type.

(c) If certificate i is self-issued and it is not the final certificate in the path, skip this step for certificate i. Otherwise, verify that the subject name is not within one of the excluded_subtrees for X.500 distinguished names, and verify that each of the alternative names in the subjectAltName extension (critical or non-critical) is not within one of the excluded_subtrees for that name type.

(d) If the certificate policies extension is present in the certificate and the valid_policy_tree is not NULL, process the policy information by performing the following steps in order:

(1) For each policy P not equal to anyPolicy in the certificate policies extension, let P-OID denote the OID in policy P and P-Q denote the qualifier set for policy P. Perform the following steps in order:

(i) If the valid_policy_tree includes a node of depth i-1 where P-OID is in the expected_policy_set, create a child node as follows: set the valid_policy to OID-P; set the qualifier_set to P-Q, and set the expected_policy_set to {P-OID}.

For example, consider a valid_policy_tree with a node of depth i-1 where the expected_policy_set is {Gold, White}. Assume the certificate policies Gold and Silver appear in the certificate policies extension of certificate i. The Gold policy is matched but the Silver policy is not. This rule will generate a child node of depth i for the Gold policy. The result is shown as Figure 4.

```
                     +-----------------+
                     |       Red       |
                     +-----------------+
                     |       {}        |
                     +-----------------+   node of depth i-1
                     |      FALSE      |
                     +-----------------+
                     |  {Gold, White}  |
                     +-----------------+
                              |
                              |
                              |
                              V
                     +-----------------+
                     |      Gold       |
                     +-----------------+
                     |       {}        |
                     +-----------------+ node of depth i
                     |  uninitialized  |
                     +-----------------+
                     |     {Gold}      |
                     +-----------------+
```

Figure 4. Processing an exact match

(ii) If there was no match in step (i) and the valid_policy_tree includes a node of depth i-1 with the valid policy anyPolicy, generate a child node with the following values: set the valid_policy to P-OID; set the qualifier_set to P-Q, and set the expected_policy_set to {P-OID}.

For example, consider a valid_policy_tree with a node of depth i-1 where the valid_policy is anyPolicy. Assume the certificate policies Gold and Silver appear in the certificate policies extension of certificate i. The Gold policy does not have a qualifier, but the Silver policy has the qualifier Q-Silver. If Gold and Silver were not matched in (i) above, this rule will generate two child nodes of depth i, one for each policy. The result is shown as Figure 5.

```
                             +-----------------+
                             |    anyPolicy    |
                             +-----------------+
                             |       {}        |
                             +-----------------+ node of depth i-1
                             |      FALSE      |
                             +-----------------+
                             |   {anyPolicy}   |
                             +-----------------+
                                /           \
                               /             \
                              /               \
                             /                 \
               +-----------------+          +-----------------+
               |      Gold       |          |     Silver      |
               +-----------------+          +-----------------+
               |       {}        |          |   {Q-Silver}    |
               +-----------------+ nodes of +-----------------+
               | uninitialized   | depth i  | uninitialized   |
               +-----------------+          +-----------------+
               |     {Gold}      |          |    {Silver}     |
               +-----------------+          +-----------------+
```

Figure 5. Processing unmatched policies when a leaf node specifies anyPolicy

(2) If the certificate policies extension includes the policy anyPolicy with the qualifier set AP-Q and either (a) inhibit_any-policy is greater than 0 or (b) i<n and the certificate is self-issued, then:

For each node in the valid_policy_tree of depth i-1, for each value in the expected_policy_set (including anyPolicy) that does not appear in a child node, create a child node with the following values: set the valid_policy to the value from the expected_policy_set in the parent node; set the qualifier_set to AP-Q, and set the expected_policy_set to the value in the valid_policy from this node.

For example, consider a valid_policy_tree with a node of depth i-1 where the expected_policy_set is {Gold, Silver}. Assume anyPolicy appears in the certificate policies extension of certificate i, but Gold and Silver do not. This rule will generate two child nodes of depth i, one for each policy. The result is shown below as Figure 6.

```
                 +-----------------+
                 |      Red        |
                 +-----------------+
                 |       {}        |
                 +-----------------+ node of depth i-1
                 |      FALSE      |
                 +-----------------+
                 |  {Gold, Silver} |
                 +-----------------+
                    /           \
                   /             \
                  /               \
                 /                 \
   +-----------------+          +-----------------+
   |      Gold       |          |     Silver      |
   +-----------------+          +-----------------+
   |       {}        |          |       {}        |
   +-----------------+ nodes of +-----------------+
   |  uninitialized  | depth i  |  uninitialized  |
   +-----------------+          +-----------------+
   |     {Gold}      |          |    {Silver}     |
   +-----------------+          +-----------------+
```

Figure 6. Processing unmatched policies when the certificate policies extension specifies anyPolicy

(3) If there is a node in the valid_policy_tree of depth i-1 or less without any child nodes, delete that node. Repeat this step until there are no nodes of depth i-1 or less without children.

For example, consider the valid_policy_tree shown in Figure 7 below. The two nodes at depth i-1 that are marked with an ’X’ have no children, and are deleted. Applying this rule to the resulting tree will cause the node at depth i-2 that is marked with an ’Y’ to be deleted. The following application of the rule does not cause any nodes to be deleted, and this step is complete.

```
                              +-----------+
                              |           | node of depth i-3
                              +-----------+
                              /     |     \
                             /      |      \
                            /       |       \
                +-----------+ +-----------+ +-----------+
                |           | |           | |     Y     | nodes of
                +-----------+ +-----------+ +-----------+ depth i-2
                /   \               |             |
               /     \              |             |
              /       \             |             |
   +-----------+ +-----------+ +-----------+ +-----------+ nodes of
   |           | |     X     | |           | |    X      |  depth
   +-----------+ +-----------+ +-----------+ +-----------+   i-1
         |                      /    |    \
         |                     /     |     \
         |                    /      |      \
   +-----------+ +-----------+ +-----------+ +-----------+ nodes of
   |           | |           | |           | |           |  depth
   +-----------+ +-----------+ +-----------+ +-----------+   i
```

Figure 7. Pruning the valid_policy_tree

(4) If the certificate policies extension was marked as critical, set the criticality_indicator in all nodes of depth i to TRUE. If the certificate policies extension was not marked critical, set the criticality_indicator in all nodes of depth i to FALSE.

(e) If the certificate policies extension is not present, set the valid_policy_tree to NULL.

(f) Verify that either explicit_policy is greater than 0 or the valid_policy_tree is not equal to NULL;

If any of steps (a), (b), (c), or (f) fails, the procedure terminates, returning a failure indication and an appropriate reason.

If i is not equal to n, continue by performing the preparatory steps listed in 6.1.4. If i is equal to n, perform the wrap-up steps listed in 6.1.5.

6.1.4 Preparation for Certificate i+1

To prepare for processing of certificate i+1, perform the following steps for certificate i:

(a) If a policy mapping extension is present, verify that the special value anyPolicy does not appear as an issuerDomainPolicy or a subjectDomainPolicy.

(b) If a policy mapping extension is present, then for each issuerDomainPolicy ID-P in the policy mapping extension:

  (1) If the policy_mapping variable is greater than 0, for each node in the valid_policy_tree of depth i where ID-P is the valid_policy, set expected_policy_set to the set of subjectDomainPolicy values that are specified as equivalent to ID-P by the policy mapping extension.

  If no node of depth i in the valid_policy_tree has a valid_policy of ID-P but there is a node of depth i with a valid_policy of anyPolicy, then generate a child node of the node of depth i-1 that has a valid_policy of anyPolicy as follows:

    (i) set the valid_policy to ID-P;

    (ii) set the qualifier_set to the qualifier set of the policy anyPolicy in the certificate policies extension of certificate i;

    (iii) set the criticality_indicator to the criticality of the certificate policies extension of certificate i;

    (iv) and set the expected_policy_set to the set of subjectDomainPolicy values that are specified as equivalent to ID-P by the policy mappings extension.

  (2) If the policy_mapping variable is equal to 0:

    (i) delete each node of depth i in the valid_policy_tree where ID-P is the valid_policy.

    (ii) If there is a node in the valid_policy_tree of depth i-1 or less without any child nodes, delete that node. Repeat this step until there are no nodes of depth i-1 or less without children.

(c) Assign the certificate subject name to working_issuer_name.

(d) Assign the certificate subjectPublicKey to working_public_key.

(e) If the subjectPublicKeyInfo field of the certificate contains an algorithm field with non-null parameters, assign the parameters to the working_public_key_parameters variable.

If the subjectPublicKeyInfo field of the certificate contains an algorithm field with null parameters or parameters are omitted, compare the certificate subjectPublicKey algorithm to the working_public_key_algorithm. If the certificate subjectPublicKey algorithm and the working_public_key_algorithm are different, set the working_public_key_parameters to null.

(f) Assign the certificate subjectPublicKey algorithm to the working_public_key_algorithm variable.

(g) If a name constraints extension is included in the certificate, modify the permitted_subtrees and excluded_subtrees state variables as follows:

(1) If permittedSubtrees is present in the certificate, set the permitted_subtrees state variable to the intersection of its previous value and the value indicated in the extension field. If permittedSubtrees does not include a particular name type, the permitted_subtrees state variable is unchanged for that name type. For example, the intersection of nist.gov and csrc.nist.gov is csrc.nist.gov. And, the intersection of nist.gov and rsasecurity.com is the empty set.

(2) If excludedSubtrees is present in the certificate, set the excluded_subtrees state variable to the union of its previous value and the value indicated in the extension field. If excludedSubtrees does not include a particular name type, the excluded_subtrees state variable is unchanged for that name type. For example, the union of the name spaces nist.gov and csrc.nist.gov is nist.gov. And, the union of nist.gov and rsasecurity.com is both name spaces.

(h) If the issuer and subject names are not identical:

(1) If explicit_policy is not 0, decrement explicit_policy by 1.

(2) If policy_mapping is not 0, decrement policy_mapping by 1.

(3) If inhibit_any-policy is not 0, decrement inhibit_any-policy by 1.

(i) If a policy constraints extension is included in the certificate, modify the explicit_policy and policy_mapping state variables as follows:

(1) If requireExplicitPolicy is present and is less than explicit_policy, set explicit_policy to the value of requireExplicitPolicy.

(2) If inhibitPolicyMapping is present and is less than policy_mapping, set policy_mapping to the value of inhibitPolicyMapping.

(j) If the inhibitAnyPolicy extension is included in the certificate and is less than inhibit_any-policy, set inhibit_any-policy to the value of inhibitAnyPolicy.

(k) Verify that the certificate is a CA certificate (as specified in a basicConstraints extension or as verified out-of-band).

(l) If the certificate was not self-issued, verify that max_path_length is greater than zero and decrement max_path_length by 1.

(m) If pathLengthConstraint is present in the certificate and is less than max_path_length, set max_path_length to the value of pathLengthConstraint.

(n) If a key usage extension is present, verify that the keyCertSign bit is set.

(o) Recognize and process any other critical extension present in the certificate. Process any other recognized non-critical extension present in the certificate.

If check (a), (k), (l), (n) or (o) fails, the procedure terminates, returning a failure indication and an appropriate reason.

If (a), (k), (l), (n) and (o) have completed successfully, increment i and perform the basic certificate processing specified in 6.1.3.

6.1.5 Wrap-up procedure

To complete the processing of the end entity certificate, perform the following steps for certificate n:

(a) If certificate n was not self-issued and explicit_policy is not 0, decrement explicit_policy by 1.

(b) If a policy constraints extension is included in the certificate and requireExplicitPolicy is present and has a value of 0, set the explicit_policy state variable to 0.

(c) Assign the certificate subjectPublicKey to working_public_key.

(d) If the subjectPublicKeyInfo field of the certificate contains an algorithm field with non-null parameters, assign the parameters to the working_public_key_parameters variable.

If the subjectPublicKeyInfo field of the certificate contains an algorithm field with null parameters or parameters are omitted, compare the certificate subjectPublicKey algorithm to the working_public_key_algorithm. If the certificate subjectPublicKey algorithm and the working_public_key_algorithm are different, set the working_public_key_parameters to null.

(e) Assign the certificate subjectPublicKey algorithm to the working_public_key_algorithm variable.

(f) Recognize and process any other critical extension present in the certificate n. Process any other recognized non-critical extension present in certificate n.

(g) Calculate the intersection of the valid_policy_tree and the user-initial-policy-set, as follows:

(i) If the valid_policy_tree is NULL, the intersection is NULL.

(ii) If the valid_policy_tree is not NULL and the user-initial-policy-set is any-policy, the intersection is the entire valid_policy_tree.

(iii) If the valid_policy_tree is not NULL and the user-initial-policy-set is not any-policy, calculate the intersection of the valid_policy_tree and the user-initial-policy-set as follows:

1. Determine the set of policy nodes whose parent nodes have a valid_policy of anyPolicy. This is the valid_policy_node_set.

2. If the valid_policy of any node in the valid_policy_node_set is not in the user-initial-policy-set and is not anyPolicy, delete this node and all its children.

3. If the valid_policy_tree includes a node of depth n with the valid_policy anyPolicy and the user-initial-policy-set is not any-policy perform the following steps:

   a. Set P-Q to the qualifier_set in the node of depth n with valid_policy anyPolicy.

   b. For each P-OID in the user-initial-policy-set that is not the valid_policy of a node in the valid_policy_node_set, create a child node whose parent is the node of depth n-1 with the valid_policy anyPolicy. Set the values in the child node as follows: set the valid_policy to P-OID; set the qualifier_set to P-Q; copy the criticality_indicator from the node of depth n with the valid_policy anyPolicy; and set the expected_policy_set to {P-OID}.

   c. Delete the node of depth n with the valid_policy anyPolicy.

4. If there is a node in the valid_policy_tree of depth n-1 or less without any child nodes, delete that node. Repeat this step until there are no nodes of depth n-1 or less without children.

If either (1) the value of explicit_policy variable is greater than zero, or (2) the valid_policy_tree is not NULL, then path processing has succeeded.

6.1.6 Outputs

If path processing succeeds, the procedure terminates, returning a success indication together with final value of the valid_policy_tree, the working_public_key, the working_public_key_algorithm, and the working_public_key_parameters.

6.2 Using the Path Validation Algorithm

The path validation algorithm describes the process of validating a single certification path. While each certification path begins with a specific trust anchor, there is no requirement that all certification paths validated by a particular system share a single trust anchor. An implementation that supports multiple trust anchors MAY augment the algorithm presented in section 6.1 to further limit the set of valid certification paths which begin with a particular trust anchor. For example, an implementation MAY modify the algorithm to apply name constraints to a specific trust anchor during the initialization phase, or the application MAY require the presence of a particular alternative name form in the end entity certificate, or the application MAY impose requirements on application-specific extensions. Thus, the path validation algorithm presented in section 6.1 defines the minimum conditions for a path to be considered valid.

The selection of one or more trusted CAs is a local decision. A system may provide any one of its trusted CAs as the trust anchor for a particular path. The inputs to the path validation algorithm may be different for each path. The inputs used to process a path may reflect application-specific requirements or limitations in the trust accorded a particular trust anchor. For example, a trusted CA may only be trusted for a particular certificate policy. This restriction can be expressed through the inputs to the path validation procedure.

It is also possible to specify an extended version of the above certification path processing procedure which results in default behavior identical to the rules of PEM [RFC 1422]. In this extended version, additional inputs to the procedure are a list of one or more Policy Certification Authority (PCA) names and an indicator of the position in the certification path where the PCA is expected. At the nominated PCA position, the CA name is compared against this list. If a recognized PCA name is found, then a constraint of SubordinateToCA is implicitly assumed for the remainder of the certification path and processing continues. If no valid PCA name is found, and if the certification path cannot be validated on the basis of identified policies, then the certification path is considered invalid.

6.3 CRL Validation

This section describes the steps necessary to determine if a certificate is revoked or on hold status when CRLs are the revocation mechanism used by the certificate issuer. Conforming implementations that support CRLs are not required to implement this algorithm, but they MUST be functionally equivalent to the external behavior resulting from this procedure. Any algorithm may be used by a particular implementation so long as it derives the correct result.

This algorithm assumes that all of the needed CRLs are available in a local cache. Further, if the next update time of a CRL has passed, the algorithm assumes a mechanism to fetch a current CRL and place it in the local CRL cache.

This algorithm defines a set of inputs, a set of state variables, and processing steps that are performed for each certificate in the path. The algorithm output is the revocation status of the certificate.

6.3.1 Revocation Inputs

To support revocation processing, the algorithm requires two inputs:

(a) certificate: The algorithm requires the certificate serial number and issuer name to determine whether a certificate is on a particular CRL. The basicConstraints extension is used to determine whether the supplied certificate is associated with a CA or an end entity. If present, the algorithm uses the cRLDistributionsPoint and freshestCRL extensions to determine revocation status.

(b) use-deltas: This boolean input determines whether delta CRLs are applied to CRLs.

Note that implementations supporting legacy PKIs, such as RFC 1422 and X.509 version 1, will need an additional input indicating whether the supplied certificate is associated with a CA or an end entity.

6.3.2 Initialization and Revocation State Variables

To support CRL processing, the algorithm requires the following state variables:

(a) reasons_mask: This variable contains the set of revocation reasons supported by the CRLs and delta CRLs processed so far. The legal members of the set are the possible revocation reason values: unspecified, keyCompromise, caCompromise, affiliationChanged, superseded, cessationOfOperation, certificateHold, privilegeWithdrawn, and aACompromise. The special value all-reasons is used to denote the set of all legal members. This variable is initialized to the empty set.

(b) cert_status: This variable contains the status of the certificate. This variable may be assigned one of the following values: unspecified, keyCompromise, caCompromise, affiliationChanged, superseded, cessationOfOperation, certificateHold, removeFromCRL, privilegeWithdrawn, aACompromise, the special value UNREVOKED, or the special value UNDETERMINED. This variable is initialized to the special value UNREVOKED.

(c) interim_reasons_mask: This contains the set of revocation reasons supported by the CRL or delta CRL currently being processed.

Note: In some environments, it is not necessary to check all reason codes. For example, some environments are only concerned with caCompromise and keyCompromise for CA certificates. This algorithm checks all reason codes. Additional processing and state variables may be necessary to limit the checking to a subset of the reason codes.

6.3.3 CRL Processing

This algorithm begins by assuming the certificate is not revoked. The algorithm checks one or more CRLs until either the certificate status is determined to be revoked or sufficient CRLs have been checked to cover all reason codes.

For each distribution point (DP) in the certificate CRL distribution points extension, for each corresponding CRL in the local CRL cache, while ((reasons_mask is not all-reasons) and (cert_status is UNREVOKED)) perform the following:

(a) Update the local CRL cache by obtaining a complete CRL, a delta CRL, or both, as required:

(1) If the current time is after the value of the CRL next update field, then do one of the following:

(i) If use-deltas is set and either the certificate or the CRL contains the freshest CRL extension, obtain a delta CRL with the a next update value that is after the current time and can be used to update the locally cached CRL as specified in section 5.2.4.

(ii) Update the local CRL cache with a current complete CRL, verify that the current time is before the next update value in the new CRL, and continue processing with the new CRL. If use-deltas is set, then obtain the current delta CRL that can be used to update the new locally cached complete CRL as specified in section 5.2.4.

(2) If the current time is before the value of the next update field and use-deltas is set, then obtain the current delta CRL that can be used to update the locally cached complete CRL as specified in section 5.2.4.

(b) Verify the issuer and scope of the complete CRL as follows:

(1) If the DP includes cRLIssuer, then verify that the issuer field in the complete CRL matches cRLIssuer in the DP and that the complete CRL contains an issuing distribution point extension with the indrectCRL boolean asserted. Otherwise, verify that the CRL issuer matches the certificate issuer.

(2) If the complete CRL includes an issuing distribution point (IDP) CRL extension check the following:

(i) If the distribution point name is present in the IDP CRL extension and the distribution field is present in the DP, then verify that one of the names in the IDP matches one of the names in the DP. If the distribution point name is present in the IDP CRL extension and the distribution field is omitted from the DP, then verify that one of the names in the IDP matches one of the names in the cRLIssuer field of the DP.

(ii) If the onlyContainsUserCerts boolean is asserted in the IDP CRL extension, verify that the certificate does not include the basic constraints extension with the cA boolean asserted.

(iii) If the onlyContainsCACerts boolean is asserted in the IDP CRL extension, verify that the certificate includes the basic constraints extension with the cA boolean asserted.

(iv) Verify that the onlyContainsAttributeCerts boolean is not asserted.

(c) If use-deltas is set, verify the issuer and scope of the delta CRL as follows:

(1) Verify that the delta CRL issuer matches complete CRL issuer.

(2) If the complete CRL includes an issuing distribution point (IDP) CRL extension, verify that the delta CRL contains a matching IDP CRL extension. If the complete CRL omits an IDP CRL extension, verify that the delta CRL also omits an IDP CRL extension.

(3) Verify that the delta CRL authority key identifier extension matches complete CRL authority key identifier extension.

(d) Compute the interim_reasons_mask for this CRL as follows:

(1) If the issuing distribution point (IDP) CRL extension is present and includes onlySomeReasons and the DP includes reasons, then set interim_reasons_mask to the intersection of reasons in the DP and onlySomeReasons in IDP CRL extension.

(2) If the IDP CRL extension includes onlySomeReasons but the DP omits reasons, then set interim_reasons_mask to the value of onlySomeReasons in IDP CRL extension.

(3) If the IDP CRL extension is not present or omits onlySomeReasons but the DP includes reasons, then set interim_reasons_mask to the value of DP reasons.

(4) If the IDP CRL extension is not present or omits onlySomeReasons and the DP omits reasons, then set interim_reasons_mask to the special value all-reasons.

(e) Verify that interim_reasons_mask includes one or more reasons that is not included in the reasons_mask.

(f) Obtain and validate the certification path for the complete CRL issuer. If a key usage extension is present in the CRL issuer’s certificate, verify that the cRLSign bit is set.

(g) Validate the signature on the complete CRL using the public key validated in step (f).

(h) If use-deltas is set, then validate the signature on the delta CRL using the public key validated in step (f).

(i) If use-deltas is set, then search for the certificate on the delta CRL. If an entry is found that matches the certificate issuer and serial number as described in section 5.3.4, then set the cert_status variable to the indicated reason as follows:

(1) If the reason code CRL entry extension is present, set the cert_status variable to the value of the reason code CRL entry extension.

(2) If the reason code CRL entry extension is not present, set the cert_status variable to the value unspecified.

(j) If (cert_status is UNREVOKED), then search for the certificate on the complete CRL. If an entry is found that matches the certificate issuer and serial number as described in section 5.3.4, then set the cert_status variable to the indicated reason as described in step (i).

(k) If (cert_status is removeFromCRL), then set cert_status to UNREVOKED.

If ((reasons_mask is all-reasons) OR (cert_status is not UNREVOKED)), then the revocation status has been determined, so return cert_status.

If the revocation status has not been determined, repeat the process above with any available CRLs not specified in a distribution point but issued by the certificate issuer. For the processing of such a CRL, assume a DP with both the reasons and the cRLIssuer fields omitted and a distribution point name of the certificate issuer. That is, the sequence of names in fullName is generated from the certificate issuer field as well as the certificate issuerAltName extension. If the revocation status remains undetermined, then return the cert_status UNDETERMINED.

7 References

[ISO 10646] ISO/IEC 10646-1:1993. International Standard -- Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.

[RFC 791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC 822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC 1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987.

[RFC 1422] Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management," RFC 1422, February 1993.

[RFC 1423] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers," RFC 1423, February 1993.

[RFC 1510] Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993.

[RFC 1519] Fuller, V., T. Li, J. Yu and K. Varadhan, "Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy", RFC 1519, September 1993.

[RFC 1738] Berners-Lee, T., L. Masinter and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[RFC 1778] Howes, T., S. Kille, W. Yeong and C. Robbins, "The String Representation of Standard Attribute Syntaxes," RFC 1778, March 1995.

[RFC 1883] Deering, S. and R. Hinden. "Internet Protocol, Version 6 (IPv6) Specification", RFC 1883, December 1995.

[RFC 2044] F. Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

[RFC 2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC 2247] Kille, S., M. Wahl, A. Grimstad, R. Huber and S. Sataluri, "Using Domains in LDAP/X.500 Distinguished Names", RFC 2247, January 1998.

[RFC 2252] Wahl, M., A. Coulbeck, T. Howes and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[RFC 2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC 2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

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

[RFC 2560] Myers, M., R. Ankney, A. Malpani, S. Galperin and C. Adams, "Online Certificate Status Protocal - OCSP", June 1999.

[SDN.701]   SDN.701, "Message Security Protocol 4.0", Revision A, 1997-02-06.

[X.501] ITU-T Recommendation X.501: Information Technology - Open Systems Interconnection - The Directory: Models, 1993.

[X.509] ITU-T Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, June 1997.

[X.520] ITU-T Recommendation X.520: Information Technology - Open Systems Interconnection - The Directory: Selected Attribute Types, 1993.

[X.660] ITU-T Recommendation X.660 Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 1997.

[X.690] ITU-T Recommendation X.690 Information Technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: General procedures, 1992.

[X9.55] ANSI X9.55-1995, Public Key Cryptography For The Financial Services Industry: Extensions To Public Key Certificates And Certificate Revocation Lists, 8 December, 1995.

[PKIXALGS] Bassham, L., Polk, W. and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation Lists (CRL) Profile", RFC 3279, April 2002.

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

8 Intellectual Property Rights

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights (see http://www.ietf.org/ipr.html).

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF’s procedures with respect to rights in standards-track and standards-related documentation can be found in BCP 11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

9 Security Considerations

The majority of this specification is devoted to the format and content of certificates and CRLs. Since certificates and CRLs are digitally signed, no additional integrity service is necessary. Neither certificates nor CRLs need be kept secret, and unrestricted and anonymous access to certificates and CRLs has no security implications.

However, security factors outside the scope of this specification will affect the assurance provided to certificate users. This section highlights critical issues to be considered by implementers, administrators, and users.

The procedures performed by CAs and RAs to validate the binding of the subject’s identity to their public key greatly affect the assurance that ought to be placed in the certificate. Relying parties might wish to review the CA’s certificate practice statement. This is particularly important when issuing certificates to other CAs.

The use of a single key pair for both signature and other purposes is strongly discouraged. Use of separate key pairs for signature and key management provides several benefits to the users. The ramifications associated with loss or disclosure of a signature key are different from loss or disclosure of a key management key. Using separate key pairs permits a balanced and flexible response. Similarly, different validity periods or key lengths for each key pair may be appropriate in some application environments. Unfortunately, some legacy applications (e.g., SSL) use a single key pair for signature and key management.

The protection afforded private keys is a critical security factor. On a small scale, failure of users to protect their private keys will permit an attacker to masquerade as them, or decrypt their personal information. On a larger scale, compromise of a CA’s private signing key may have a catastrophic effect. If an attacker obtains the private key unnoticed, the attacker may issue bogus certificates and CRLs. Existence of bogus certificates and CRLs will undermine confidence in the system. If such a compromise is detected, all certificates issued to the compromised CA MUST be revoked, preventing services between its users and users of other CAs. Rebuilding after such a compromise will be problematic, so CAs are advised to implement a combination of strong technical measures (e.g., tamper-resistant cryptographic modules) and appropriate management procedures (e.g., separation of duties) to avoid such an incident.

Loss of a CA’s private signing key may also be problematic. The CA would not be able to produce CRLs or perform normal key rollover. CAs SHOULD maintain secure backup for signing keys. The security of the key backup procedures is a critical factor in avoiding key compromise.

The availability and freshness of revocation information affects the degree of assurance that ought to be placed in a certificate. While certificates expire naturally, events may occur during its natural lifetime which negate the binding between the subject and public key. If revocation information is untimely or unavailable, the assurance associated with the binding is clearly reduced. Relying parties might not be able to process every critical extension that can appear in a CRL. CAs SHOULD take extra care when making revocation information available only through CRLs that contain critical extensions, particularly if support for those extensions is not mandated by this profile. For example, if revocation information is supplied using a combination of delta CRLs and full CRLs, and the delta CRLs are issued more frequently than the full CRLs, then relying parties that cannot handle the critical extensions related to delta CRL processing will not be able to obtain the most recent revocation information. Alternatively, if a full CRL is issued whenever a delta CRL is issued, then timely revocation information will be available to all relying parties. Similarly, implementations of the certification path validation mechanism described in section 6 that omit revocation checking provide less assurance than those that support it.

The certification path validation algorithm depends on the certain knowledge of the public keys (and other information) about one or more trusted CAs. The decision to trust a CA is an important decision as it ultimately determines the trust afforded a certificate. The authenticated distribution of trusted CA public keys (usually in the form of a "self-signed" certificate) is a security critical out-of-band process that is beyond the scope of this specification.

In addition, where a key compromise or CA failure occurs for a trusted CA, the user will need to modify the information provided to the path validation routine. Selection of too many trusted CAs makes the trusted CA information difficult to maintain. On the other hand, selection of only one trusted CA could limit users to a closed community of users.

The quality of implementations that process certificates also affects the degree of assurance provided. The path validation algorithm described in section 6 relies upon the integrity of the trusted CA information, and especially the integrity of the public keys associated with the trusted CAs. By substituting public keys for which an attacker has the private key, an attacker could trick the user into accepting false certificates.

The binding between a key and certificate subject cannot be stronger than the cryptographic module implementation and algorithms used to generate the signature. Short key lengths or weak hash algorithms will limit the utility of a certificate. CAs are encouraged to note advances in cryptology so they can employ strong cryptographic techniques. In addition, CAs SHOULD decline to issue certificates to CAs or end entities that generate weak signatures.

Inconsistent application of name comparison rules can result in acceptance of invalid X.509 certification paths, or rejection of valid ones. The X.500 series of specifications defines rules for comparing distinguished names that require comparison of strings without regard to case, character set, multi-character white space substring, or leading and trailing white space. This specification relaxes these requirements, requiring support for binary comparison at a minimum.

CAs MUST encode the distinguished name in the subject field of a CA certificate identically to the distinguished name in the issuer field in certificates issued by that CA. If CAs use different encodings, implementations might fail to recognize name chains for paths that include this certificate. As a consequence, valid paths could be rejected.

In addition, name constraints for distinguished names MUST be stated identically to the encoding used in the subject field or subjectAltName extension. If not, then name constraints stated as excludedSubTrees will not match and invalid paths will be accepted and name constraints expressed as permittedSubtrees will not match and valid paths will be rejected. To avoid acceptance of invalid paths, CAs SHOULD state name constraints for distinguished names as permittedSubtrees wherever possible.

Appendix A. Psuedo-ASN.1 Structures and OIDs

This section describes data objects used by conforming PKI components in an "ASN.1-like" syntax. This syntax is a hybrid of the 1988 and 1993 ASN.1 syntaxes. The 1988 ASN.1 syntax is augmented with 1993 UNIVERSAL Types UniversalString, BMPString and UTF8String.

The ASN.1 syntax does not permit the inclusion of type statements in the ASN.1 module, and the 1993 ASN.1 standard does not permit use of the new UNIVERSAL types in modules using the 1988 syntax. As a result, this module does not conform to either version of the ASN.1 standard.

This appendix may be converted into 1988 ASN.1 by replacing the definitions for the UNIVERSAL Types with the 1988 catch-all "ANY".

A.1 Explicitly Tagged Module, 1988 Syntax

```
PKIX1Explicit88 { iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit(18) }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

-- IMPORTS NONE --

-- UNIVERSAL Types defined in 1993 and 1998 ASN.1
-- and required by this specification

UniversalString ::= [UNIVERSAL 28] IMPLICIT OCTET STRING
        -- UniversalString is defined in ASN.1:1993

BMPString ::= [UNIVERSAL 30] IMPLICIT OCTET STRING
      -- BMPString is the subtype of UniversalString and models
      -- the Basic Multilingual Plane of ISO/IEC/ITU 10646-1

UTF8String ::= [UNIVERSAL 12] IMPLICIT OCTET STRING
      -- The content of this type conforms to RFC 2279.

-- PKIX specific OIDs

id-pkix  OBJECT IDENTIFIER  ::=
         { iso(1) identified-organization(3) dod(6) internet(1)
                    security(5) mechanisms(5) pkix(7) }
```

```
-- PKIX arcs

id-pe OBJECT IDENTIFIER  ::=  { id-pkix 1 }
        -- arc for private certificate extensions
id-qt OBJECT IDENTIFIER ::= { id-pkix 2 }
        -- arc for policy qualifier types
id-kp OBJECT IDENTIFIER ::= { id-pkix 3 }
        -- arc for extended key purpose OIDS
id-ad OBJECT IDENTIFIER ::= { id-pkix 48 }
        -- arc for access descriptors

-- policyQualifierIds for Internet policy qualifiers

id-qt-cps      OBJECT IDENTIFIER ::=  { id-qt 1 }
      -- OID for CPS qualifier
id-qt-unotice  OBJECT IDENTIFIER ::=  { id-qt 2 }
      -- OID for user notice qualifier

-- access descriptor definitions

id-ad-ocsp         OBJECT IDENTIFIER ::= { id-ad 1 }
id-ad-caIssuers    OBJECT IDENTIFIER ::= { id-ad 2 }
id-ad-timeStamping OBJECT IDENTIFIER ::= { id-ad 3 }
id-ad-caRepository OBJECT IDENTIFIER ::= { id-ad 5 }

-- attribute data types

Attribute       ::=     SEQUENCE {
      type             AttributeType,
      values    SET OF AttributeValue }
            -- at least one value is required

AttributeType           ::=  OBJECT IDENTIFIER

AttributeValue          ::=  ANY

AttributeTypeAndValue           ::=     SEQUENCE {
        type    AttributeType,
        value   AttributeValue }

-- suggested naming attributes: Definition of the following
--   information object set may be augmented to meet local
--   requirements.  Note that deleting members of the set may
--   prevent interoperability with conforming implementations.
-- presented in pairs: the AttributeType followed by the
--   type definition for the corresponding AttributeValue
--Arc for standard naming attributes
id-at OBJECT IDENTIFIER ::= { joint-iso-ccitt(2) ds(5) 4 }
```

```
-- Naming attributes of type X520name

id-at-name              AttributeType ::= { id-at 41 }
id-at-surname           AttributeType ::= { id-at 4 }
id-at-givenName         AttributeType ::= { id-at 42 }
id-at-initials          AttributeType ::= { id-at 43 }
id-at-generationQualifier AttributeType ::= { id-at 44 }

X520name ::= CHOICE {
      teletexString     TeletexString   (SIZE (1..ub-name)),
      printableString   PrintableString (SIZE (1..ub-name)),
      universalString   UniversalString (SIZE (1..ub-name)),
      utf8String        UTF8String      (SIZE (1..ub-name)),
      bmpString         BMPString       (SIZE (1..ub-name)) }

-- Naming attributes of type X520CommonName

id-at-commonName        AttributeType ::= { id-at 3 }

X520CommonName ::= CHOICE {
      teletexString     TeletexString   (SIZE (1..ub-common-name)),
      printableString   PrintableString (SIZE (1..ub-common-name)),
      universalString   UniversalString (SIZE (1..ub-common-name)),
      utf8String        UTF8String      (SIZE (1..ub-common-name)),
      bmpString         BMPString       (SIZE (1..ub-common-name)) }

-- Naming attributes of type X520LocalityName

id-at-localityName      AttributeType ::= { id-at 7 }

X520LocalityName ::= CHOICE {
      teletexString     TeletexString   (SIZE (1..ub-locality-name)),
      printableString   PrintableString (SIZE (1..ub-locality-name)),
      universalString   UniversalString (SIZE (1..ub-locality-name)),
      utf8String        UTF8String      (SIZE (1..ub-locality-name)),
      bmpString         BMPString       (SIZE (1..ub-locality-name)) }

-- Naming attributes of type X520StateOrProvinceName

id-at-stateOrProvinceName AttributeType ::= { id-at 8 }

X520StateOrProvinceName ::= CHOICE {
      teletexString     TeletexString   (SIZE (1..ub-state-name)),
      printableString   PrintableString (SIZE (1..ub-state-name)),
      universalString   UniversalString (SIZE (1..ub-state-name)),
      utf8String        UTF8String      (SIZE (1..ub-state-name)),
      bmpString         BMPString       (SIZE(1..ub-state-name)) }
```

```
-- Naming attributes of type X520OrganizationName

id-at-organizationName  AttributeType ::= { id-at 10 }

X520OrganizationName ::= CHOICE {
      teletexString     TeletexString
                          (SIZE (1..ub-organization-name)),
      printableString   PrintableString
                          (SIZE (1..ub-organization-name)),
      universalString   UniversalString
                          (SIZE (1..ub-organization-name)),
      utf8String        UTF8String
                          (SIZE (1..ub-organization-name)),
      bmpString         BMPString
                          (SIZE (1..ub-organization-name))  }

-- Naming attributes of type X520OrganizationalUnitName

id-at-organizationalUnitName AttributeType ::= { id-at 11 }

X520OrganizationalUnitName ::= CHOICE {
      teletexString     TeletexString
                          (SIZE (1..ub-organizational-unit-name)),
      printableString   PrintableString
                          (SIZE (1..ub-organizational-unit-name)),
      universalString   UniversalString
                          (SIZE (1..ub-organizational-unit-name)),
      utf8String        UTF8String
                          (SIZE (1..ub-organizational-unit-name)),
      bmpString         BMPString
                          (SIZE (1..ub-organizational-unit-name)) }

-- Naming attributes of type X520Title

id-at-title             AttributeType ::= { id-at 12 }

X520Title ::= CHOICE {
      teletexString     TeletexString   (SIZE (1..ub-title)),
      printableString   PrintableString (SIZE (1..ub-title)),
      universalString   UniversalString (SIZE (1..ub-title)),
      utf8String        UTF8String      (SIZE (1..ub-title)),
      bmpString         BMPString       (SIZE (1..ub-title)) }

-- Naming attributes of type X520dnQualifier

id-at-dnQualifier       AttributeType ::= { id-at 46 }

X520dnQualifier ::=     PrintableString
```

```
-- Naming attributes of type X520countryName (digraph from IS 3166)

id-at-countryName       AttributeType ::= { id-at 6 }

X520countryName ::=     PrintableString (SIZE (2))

-- Naming attributes of type X520SerialNumber

id-at-serialNumber      AttributeType ::= { id-at 5 }

X520SerialNumber ::=    PrintableString (SIZE (1..ub-serial-number))

-- Naming attributes of type X520Pseudonym

id-at-pseudonym         AttributeType ::= { id-at 65 }

X520Pseudonym ::= CHOICE {
   teletexString     TeletexString   (SIZE (1..ub-pseudonym)),
   printableString   PrintableString (SIZE (1..ub-pseudonym)),
   universalString   UniversalString (SIZE (1..ub-pseudonym)),
   utf8String        UTF8String      (SIZE (1..ub-pseudonym)),
   bmpString         BMPString       (SIZE (1..ub-pseudonym)) }

-- Naming attributes of type DomainComponent (from RFC 2247)

id-domainComponent      AttributeType ::=
                          { 0 9 2342 19200300 100 1 25 }

DomainComponent ::=     IA5String

-- Legacy attributes

pkcs-9 OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 9 }

id-emailAddress          AttributeType ::= { pkcs-9 1 }

EmailAddress ::=         IA5String (SIZE (1..ub-emailaddress-length))

-- naming data types --

Name ::= CHOICE { -- only one possibility for now --
      rdnSequence  RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

DistinguishedName ::=   RDNSequence
```

```
RelativeDistinguishedName  ::=
                    SET SIZE (1 .. MAX) OF AttributeTypeAndValue

-- Directory string type --

DirectoryString ::= CHOICE {
      teletexString             TeletexString   (SIZE (1..MAX)),
      printableString           PrintableString (SIZE (1..MAX)),
      universalString           UniversalString (SIZE (1..MAX)),
      utf8String              UTF8String      (SIZE (1..MAX)),
      bmpString               BMPString       (SIZE (1..MAX)) }

-- certificate and CRL specific structures begin here

Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signature            BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  Extensions OPTIONAL
                          -- If present, version MUST be v3 --  }

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

CertificateSerialNumber  ::=  INTEGER

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

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

UniqueIdentifier  ::=  BIT STRING
```

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

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING  }

-- CRL structures

CertificateList  ::=  SEQUENCE  {
     tbsCertList          TBSCertList,
     signatureAlgorithm   AlgorithmIdentifier,
     signature            BIT STRING  }

TBSCertList  ::=  SEQUENCE  {
     version                 Version OPTIONAL,
                                  -- if present, MUST be v2
     signature               AlgorithmIdentifier,
     issuer                  Name,
     thisUpdate              Time,
     nextUpdate              Time OPTIONAL,
     revokedCertificates     SEQUENCE OF SEQUENCE  {
          userCertificate         CertificateSerialNumber,
          revocationDate          Time,
          crlEntryExtensions      Extensions OPTIONAL
                                         -- if present, MUST be v2
                               }  OPTIONAL,
     crlExtensions           [0] Extensions OPTIONAL }
                                         -- if present, MUST be v2

-- Version, Time, CertificateSerialNumber, and Extensions were
-- defined earlier for use in the certificate structure

AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }
                                -- contains a value of the type
                                -- registered for use with the
                                -- algorithm object identifier value

-- X.400 address syntax starts here
```

```
ORAddress ::= SEQUENCE {
   built-in-standard-attributes BuiltInStandardAttributes,
   built-in-domain-defined-attributes
                   BuiltInDomainDefinedAttributes OPTIONAL,
   -- see also teletex-domain-defined-attributes
   extension-attributes ExtensionAttributes OPTIONAL }

-- Built-in Standard Attributes

BuiltInStandardAttributes ::= SEQUENCE {
   country-name                  CountryName OPTIONAL,
   administration-domain-name    AdministrationDomainName OPTIONAL,
   network-address           [0] IMPLICIT NetworkAddress OPTIONAL,
     -- see also extended-network-address
   terminal-identifier       [1] IMPLICIT TerminalIdentifier OPTIONAL,
   private-domain-name       [2] PrivateDomainName OPTIONAL,
   organization-name         [3] IMPLICIT OrganizationName OPTIONAL,
     -- see also teletex-organization-name
   numeric-user-identifier   [4] IMPLICIT NumericUserIdentifier
                                 OPTIONAL,
   personal-name             [5] IMPLICIT PersonalName OPTIONAL,
     -- see also teletex-personal-name
   organizational-unit-names [6] IMPLICIT OrganizationalUnitNames
                                 OPTIONAL }
     -- see also teletex-organizational-unit-names

CountryName ::= [APPLICATION 1] CHOICE {
   x121-dcc-code         NumericString
                           (SIZE (ub-country-name-numeric-length)),
   iso-3166-alpha2-code  PrintableString
                           (SIZE (ub-country-name-alpha-length)) }

AdministrationDomainName ::= [APPLICATION 2] CHOICE {
   numeric   NumericString   (SIZE (0..ub-domain-name-length)),
   printable PrintableString (SIZE (0..ub-domain-name-length)) }

NetworkAddress ::= X121Address  -- see also extended-network-address

X121Address ::= NumericString (SIZE (1..ub-x121-address-length))

TerminalIdentifier ::= PrintableString (SIZE
(1..ub-terminal-id-length))

PrivateDomainName ::= CHOICE {
   numeric   NumericString   (SIZE (1..ub-domain-name-length)),
   printable PrintableString (SIZE (1..ub-domain-name-length)) }
```

```
OrganizationName ::= PrintableString
                            (SIZE (1..ub-organization-name-length))
  -- see also teletex-organization-name

NumericUserIdentifier ::= NumericString
                            (SIZE (1..ub-numeric-user-id-length))

PersonalName ::= SET {
   surname     [0] IMPLICIT PrintableString
                    (SIZE (1..ub-surname-length)),
   given-name  [1] IMPLICIT PrintableString
                    (SIZE (1..ub-given-name-length)) OPTIONAL,
   initials    [2] IMPLICIT PrintableString
                    (SIZE (1..ub-initials-length)) OPTIONAL,
   generation-qualifier [3] IMPLICIT PrintableString
                    (SIZE (1..ub-generation-qualifier-length))
                    OPTIONAL }
  -- see also teletex-personal-name

OrganizationalUnitNames ::= SEQUENCE SIZE (1..ub-organizational-units)
                             OF OrganizationalUnitName
  -- see also teletex-organizational-unit-names

OrganizationalUnitName ::= PrintableString (SIZE
                    (1..ub-organizational-unit-name-length))

-- Built-in Domain-defined Attributes

BuiltInDomainDefinedAttributes ::= SEQUENCE SIZE
                    (1..ub-domain-defined-attributes) OF
                    BuiltInDomainDefinedAttribute

BuiltInDomainDefinedAttribute ::= SEQUENCE {
   type PrintableString (SIZE
                   (1..ub-domain-defined-attribute-type-length)),
   value PrintableString (SIZE
                   (1..ub-domain-defined-attribute-value-length)) }

-- Extension Attributes

ExtensionAttributes ::= SET SIZE (1..ub-extension-attributes) OF
               ExtensionAttribute

ExtensionAttribute ::=  SEQUENCE {
   extension-attribute-type [0] IMPLICIT INTEGER
                   (0..ub-extension-attributes),
   extension-attribute-value [1]
                   ANY DEFINED BY extension-attribute-type }
```

```
-- Extension types and attribute values

common-name INTEGER ::= 1

CommonName ::= PrintableString (SIZE (1..ub-common-name-length))

teletex-common-name INTEGER ::= 2

TeletexCommonName ::= TeletexString (SIZE (1..ub-common-name-length))

teletex-organization-name INTEGER ::= 3

TeletexOrganizationName ::=
                TeletexString (SIZE (1..ub-organization-name-length))

teletex-personal-name INTEGER ::= 4

TeletexPersonalName ::= SET {
   surname     [0] IMPLICIT TeletexString
                    (SIZE (1..ub-surname-length)),
   given-name  [1] IMPLICIT TeletexString
                    (SIZE (1..ub-given-name-length)) OPTIONAL,
   initials    [2] IMPLICIT TeletexString
                    (SIZE (1..ub-initials-length)) OPTIONAL,
   generation-qualifier [3] IMPLICIT TeletexString
                    (SIZE (1..ub-generation-qualifier-length))
                    OPTIONAL }

teletex-organizational-unit-names INTEGER ::= 5

TeletexOrganizationalUnitNames ::= SEQUENCE SIZE
      (1..ub-organizational-units) OF TeletexOrganizationalUnitName

TeletexOrganizationalUnitName ::= TeletexString
                  (SIZE (1..ub-organizational-unit-name-length))

pds-name INTEGER ::= 7

PDSName ::= PrintableString (SIZE (1..ub-pds-name-length))

physical-delivery-country-name INTEGER ::= 8

PhysicalDeliveryCountryName ::= CHOICE {
   x121-dcc-code NumericString (SIZE
(ub-country-name-numeric-length)),
   iso-3166-alpha2-code PrintableString
                  (SIZE (ub-country-name-alpha-length)) }
```

```
postal-code INTEGER ::= 9

PostalCode ::= CHOICE {
   numeric-code NumericString (SIZE (1..ub-postal-code-length)),
   printable-code PrintableString (SIZE (1..ub-postal-code-length)) }

physical-delivery-office-name INTEGER ::= 10

PhysicalDeliveryOfficeName ::= PDSParameter

physical-delivery-office-number INTEGER ::= 11

PhysicalDeliveryOfficeNumber ::= PDSParameter

extension-OR-address-components INTEGER ::= 12

ExtensionORAddressComponents ::= PDSParameter

physical-delivery-personal-name INTEGER ::= 13

PhysicalDeliveryPersonalName ::= PDSParameter

physical-delivery-organization-name INTEGER ::= 14

PhysicalDeliveryOrganizationName ::= PDSParameter

extension-physical-delivery-address-components INTEGER ::= 15

ExtensionPhysicalDeliveryAddressComponents ::= PDSParameter

unformatted-postal-address INTEGER ::= 16

UnformattedPostalAddress ::= SET {
   printable-address SEQUENCE SIZE (1..ub-pds-physical-address-lines)
         OF PrintableString (SIZE (1..ub-pds-parameter-length))
         OPTIONAL,
   teletex-string TeletexString
         (SIZE (1..ub-unformatted-address-length)) OPTIONAL }

street-address INTEGER ::= 17

StreetAddress ::= PDSParameter

post-office-box-address INTEGER ::= 18

PostOfficeBoxAddress ::= PDSParameter

poste-restante-address INTEGER ::= 19
```

```
PosteRestanteAddress ::= PDSParameter

unique-postal-name INTEGER ::= 20

UniquePostalName ::= PDSParameter

local-postal-attributes INTEGER ::= 21

LocalPostalAttributes ::= PDSParameter

PDSParameter ::= SET {
   printable-string PrintableString
                (SIZE(1..ub-pds-parameter-length)) OPTIONAL,
   teletex-string TeletexString
                (SIZE(1..ub-pds-parameter-length)) OPTIONAL }

extended-network-address INTEGER ::= 22

ExtendedNetworkAddress ::= CHOICE {
   e163-4-address SEQUENCE {
      number      [0] IMPLICIT NumericString
                       (SIZE (1..ub-e163-4-number-length)),
      sub-address [1] IMPLICIT NumericString
                       (SIZE (1..ub-e163-4-sub-address-length))
                       OPTIONAL },
   psap-address [0] IMPLICIT PresentationAddress }

PresentationAddress ::= SEQUENCE {
    pSelector     [0] EXPLICIT OCTET STRING OPTIONAL,
    sSelector     [1] EXPLICIT OCTET STRING OPTIONAL,
    tSelector     [2] EXPLICIT OCTET STRING OPTIONAL,
    nAddresses    [3] EXPLICIT SET SIZE (1..MAX) OF OCTET STRING }

terminal-type  INTEGER ::= 23

TerminalType ::= INTEGER {
   telex (3),
   teletex (4),
   g3-facsimile (5),
   g4-facsimile (6),
   ia5-terminal (7),
   videotex (8) } (0..ub-integer-options)

-- Extension Domain-defined Attributes

teletex-domain-defined-attributes INTEGER ::= 6
```

```
TeletexDomainDefinedAttributes ::= SEQUENCE SIZE
   (1..ub-domain-defined-attributes) OF TeletexDomainDefinedAttribute

TeletexDomainDefinedAttribute ::= SEQUENCE {
        type TeletexString
               (SIZE (1..ub-domain-defined-attribute-type-length)),
        value TeletexString
               (SIZE (1..ub-domain-defined-attribute-value-length)) }

--  specifications of Upper Bounds MUST be regarded as mandatory
--  from Annex B of ITU-T X.411 Reference Definition of MTS Parameter
--  Upper Bounds

-- Upper Bounds
ub-name INTEGER ::= 32768
ub-common-name INTEGER ::= 64
ub-locality-name INTEGER ::= 128
ub-state-name INTEGER ::= 128
ub-organization-name INTEGER ::= 64
ub-organizational-unit-name INTEGER ::= 64
ub-title INTEGER ::= 64
ub-serial-number INTEGER ::= 64
ub-match INTEGER ::= 128
ub-emailaddress-length INTEGER ::= 128
ub-common-name-length INTEGER ::= 64
ub-country-name-alpha-length INTEGER ::= 2
ub-country-name-numeric-length INTEGER ::= 3
ub-domain-defined-attributes INTEGER ::= 4
ub-domain-defined-attribute-type-length INTEGER ::= 8
ub-domain-defined-attribute-value-length INTEGER ::= 128
ub-domain-name-length INTEGER ::= 16
ub-extension-attributes INTEGER ::= 256
ub-e163-4-number-length INTEGER ::= 15
ub-e163-4-sub-address-length INTEGER ::= 40
ub-generation-qualifier-length INTEGER ::= 3
ub-given-name-length INTEGER ::= 16
ub-initials-length INTEGER ::= 5
ub-integer-options INTEGER ::= 256
ub-numeric-user-id-length INTEGER ::= 32
ub-organization-name-length INTEGER ::= 64
ub-organizational-unit-name-length INTEGER ::= 32
ub-organizational-units INTEGER ::= 4
ub-pds-name-length INTEGER ::= 16
ub-pds-parameter-length INTEGER ::= 30
ub-pds-physical-address-lines INTEGER ::= 6
ub-postal-code-length INTEGER ::= 16
ub-pseudonym INTEGER ::= 128
ub-surname-length INTEGER ::= 40
```

```
ub-terminal-id-length INTEGER ::= 24
ub-unformatted-address-length INTEGER ::= 180
ub-x121-address-length INTEGER ::= 16

-- Note - upper bounds on string types, such as TeletexString, are
-- measured in characters.  Excepting PrintableString or IA5String, a
-- significantly greater number of octets will be required to hold
-- such a value.  As a minimum, 16 octets, or twice the specified
-- upper bound, whichever is the larger, should be allowed for
-- TeletexString.  For UTF8String or UniversalString at least four
-- times the upper bound should be allowed.

END
```

A.2 Implicitly Tagged Module, 1988 Syntax

```
PKIX1Implicit88 { iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit(19) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

IMPORTS
      id-pe, id-kp, id-qt-unotice, id-qt-cps,
      -- delete following line if "new" types are supported --
      BMPString, UTF8String,  -- end "new" types --
      ORAddress, Name, RelativeDistinguishedName,
      CertificateSerialNumber, Attribute, DirectoryString
      FROM PKIX1Explicit88 { iso(1) identified-organization(3)
            dod(6) internet(1) security(5) mechanisms(5) pkix(7)
            id-mod(0) id-pkix1-explicit(18) };


-- ISO arc for standard certificate and CRL extensions

id-ce OBJECT IDENTIFIER  ::=  {joint-iso-ccitt(2) ds(5) 29}

-- authority key identifier OID and syntax

id-ce-authorityKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 35 }
```

```
AuthorityKeyIdentifier ::= SEQUENCE {
    keyIdentifier             [0] KeyIdentifier            OPTIONAL,
    authorityCertIssuer       [1] GeneralNames             OPTIONAL,
    authorityCertSerialNumber [2] CertificateSerialNumber  OPTIONAL }
    -- authorityCertIssuer and authorityCertSerialNumber MUST both
    -- be present or both be absent

KeyIdentifier ::= OCTET STRING

-- subject key identifier OID and syntax

id-ce-subjectKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 14 }

SubjectKeyIdentifier ::= KeyIdentifier

-- key usage extension OID and syntax

id-ce-keyUsage OBJECT IDENTIFIER ::=  { id-ce 15 }

KeyUsage ::= BIT STRING {
     digitalSignature        (0),
     nonRepudiation          (1),
     keyEncipherment         (2),
     dataEncipherment        (3),
     keyAgreement            (4),
     keyCertSign             (5),
     cRLSign                 (6),
     encipherOnly            (7),
     decipherOnly            (8) }

-- private key usage period extension OID and syntax

id-ce-privateKeyUsagePeriod OBJECT IDENTIFIER ::=  { id-ce 16 }

PrivateKeyUsagePeriod ::= SEQUENCE {
     notBefore       [0]     GeneralizedTime OPTIONAL,
     notAfter        [1]     GeneralizedTime OPTIONAL }
     -- either notBefore or notAfter MUST be present

-- certificate policies extension OID and syntax

id-ce-certificatePolicies OBJECT IDENTIFIER ::=  { id-ce 32 }

anyPolicy OBJECT IDENTIFIER ::= { id-ce-certificatePolicies 0 }

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

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

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
       policyQualifierId  PolicyQualifierId,
       qualifier        ANY DEFINED BY policyQualifierId }

-- Implementations that recognize additional policy qualifiers MUST
-- augment the following definition for PolicyQualifierId

PolicyQualifierId ::=
    OBJECT IDENTIFIER ( id-qt-cps | id-qt-unotice )

-- CPS pointer qualifier

CPSuri ::= IA5String

-- user notice qualifier

UserNotice ::= SEQUENCE {
     noticeRef        NoticeReference OPTIONAL,
     explicitText     DisplayText OPTIONAL}

NoticeReference ::= SEQUENCE {
     organization     DisplayText,
     noticeNumbers    SEQUENCE OF INTEGER }

DisplayText ::= CHOICE {
     ia5String        IA5String      (SIZE (1..200)),
     visibleString    VisibleString  (SIZE (1..200)),
     bmpString        BMPString      (SIZE (1..200)),
     utf8String       UTF8String     (SIZE (1..200)) }

-- policy mapping extension OID and syntax

id-ce-policyMappings OBJECT IDENTIFIER ::=  { id-ce 33 }

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

-- subject alternative name extension OID and syntax

id-ce-subjectAltName OBJECT IDENTIFIER ::=  { id-ce 17 }
```

```
SubjectAltName ::= GeneralNames

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

GeneralName ::= CHOICE {
     otherName                       [0]     AnotherName,
     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 }

-- AnotherName replaces OTHER-NAME ::= TYPE-IDENTIFIER, as
-- TYPE-IDENTIFIER is not supported in the ’88 ASN.1 syntax

AnotherName ::= SEQUENCE {
     type-id    OBJECT IDENTIFIER,
     value      [0] EXPLICIT ANY DEFINED BY type-id }

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

-- issuer alternative name extension OID and syntax

id-ce-issuerAltName OBJECT IDENTIFIER ::=  { id-ce 18 }

IssuerAltName ::= GeneralNames

id-ce-subjectDirectoryAttributes OBJECT IDENTIFIER ::=  { id-ce 9 }

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

-- basic constraints extension OID and syntax

id-ce-basicConstraints OBJECT IDENTIFIER ::=  { id-ce 19 }

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

-- name constraints extension OID and syntax

id-ce-nameConstraints OBJECT IDENTIFIER ::=  { id-ce 30 }
```

```
NameConstraints ::= 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)

-- policy constraints extension OID and syntax

id-ce-policyConstraints OBJECT IDENTIFIER ::=  { id-ce 36 }

PolicyConstraints ::= SEQUENCE {
     requireExplicitPolicy           [0] SkipCerts OPTIONAL,
     inhibitPolicyMapping            [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)

-- CRL distribution points extension OID and syntax

id-ce-cRLDistributionPoints     OBJECT IDENTIFIER  ::=  {id-ce 31}

CRLDistributionPoints ::= 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),
     privilegeWithdrawn      (7),
     aACompromise            (8) }
```

```
-- extended key usage extension OID and syntax

id-ce-extKeyUsage OBJECT IDENTIFIER ::= {id-ce 37}

ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId


KeyPurposeId ::= OBJECT IDENTIFIER

-- permit unspecified key uses

anyExtendedKeyUsage OBJECT IDENTIFIER ::= { id-ce-extKeyUsage 0 }

-- extended key purpose OIDs

id-kp-serverAuth             OBJECT IDENTIFIER ::= { id-kp 1 }
id-kp-clientAuth             OBJECT IDENTIFIER ::= { id-kp 2 }
id-kp-codeSigning            OBJECT IDENTIFIER ::= { id-kp 3 }
id-kp-emailProtection        OBJECT IDENTIFIER ::= { id-kp 4 }
id-kp-timeStamping           OBJECT IDENTIFIER ::= { id-kp 8 }
id-kp-OCSPSigning            OBJECT IDENTIFIER ::= { id-kp 9 }

-- inhibit any policy OID and syntax

id-ce-inhibitAnyPolicy OBJECT IDENTIFIER ::=  { id-ce 54 }

InhibitAnyPolicy ::= SkipCerts

-- freshest (delta)CRL extension OID and syntax

id-ce-freshestCRL OBJECT IDENTIFIER ::=  { id-ce 46 }

FreshestCRL ::= CRLDistributionPoints

-- authority info access

id-pe-authorityInfoAccess OBJECT IDENTIFIER ::= { id-pe 1 }

AuthorityInfoAccessSyntax  ::=
        SEQUENCE SIZE (1..MAX) OF AccessDescription

AccessDescription  ::=  SEQUENCE {
        accessMethod          OBJECT IDENTIFIER,
        accessLocation        GeneralName  }

-- subject info access

id-pe-subjectInfoAccess OBJECT IDENTIFIER ::= { id-pe 11 }
```

```
SubjectInfoAccessSyntax  ::=
        SEQUENCE SIZE (1..MAX) OF AccessDescription

-- CRL number extension OID and syntax

id-ce-cRLNumber OBJECT IDENTIFIER ::= { id-ce 20 }

CRLNumber ::= INTEGER (0..MAX)

-- issuing distribution point extension OID and syntax

id-ce-issuingDistributionPoint OBJECT IDENTIFIER ::= { id-ce 28 }

IssuingDistributionPoint ::= SEQUENCE {
     distributionPoint          [0] DistributionPointName OPTIONAL,
     onlyContainsUserCerts      [1] BOOLEAN DEFAULT FALSE,
     onlyContainsCACerts        [2] BOOLEAN DEFAULT FALSE,
     onlySomeReasons            [3] ReasonFlags OPTIONAL,
     indirectCRL                [4] BOOLEAN DEFAULT FALSE,
     onlyContainsAttributeCerts [5] BOOLEAN DEFAULT FALSE }

id-ce-deltaCRLIndicator OBJECT IDENTIFIER ::= { id-ce 27 }

BaseCRLNumber ::= CRLNumber

-- CRL reasons extension OID and syntax

id-ce-cRLReasons OBJECT IDENTIFIER ::= { id-ce 21 }

CRLReason ::= ENUMERATED {
     unspecified             (0),
     keyCompromise           (1),
     cACompromise            (2),
     affiliationChanged      (3),
     superseded              (4),
     cessationOfOperation    (5),
     certificateHold         (6),
     removeFromCRL           (8),
     privilegeWithdrawn      (9),
     aACompromise           (10) }

-- certificate issuer CRL entry extension OID and syntax

id-ce-certificateIssuer OBJECT IDENTIFIER ::= { id-ce 29 }

CertificateIssuer ::= GeneralNames

-- hold instruction extension OID and syntax
```

```
id-ce-holdInstructionCode OBJECT IDENTIFIER ::= { id-ce 23 }

HoldInstructionCode ::= OBJECT IDENTIFIER

-- ANSI x9 holdinstructions

-- ANSI x9 arc holdinstruction arc

holdInstruction OBJECT IDENTIFIER ::=
          {joint-iso-itu-t(2) member-body(2) us(840) x9cm(10040) 2}

-- ANSI X9 holdinstructions referenced by this standard

id-holdinstruction-none OBJECT IDENTIFIER  ::=
                {holdInstruction 1} -- deprecated

id-holdinstruction-callissuer OBJECT IDENTIFIER ::=
                {holdInstruction 2}

id-holdinstruction-reject OBJECT IDENTIFIER ::=
                {holdInstruction 3}

-- invalidity date CRL entry extension OID and syntax

id-ce-invalidityDate OBJECT IDENTIFIER ::= { id-ce 24 }

InvalidityDate ::=  GeneralizedTime

END
```

Appendix B. ASN.1 Notes

CAs MUST force the serialNumber to be a non-negative integer, that is, the sign bit in the DER encoding of the INTEGER value MUST be zero - this can be done by adding a leading (leftmost) ‘00’H octet if necessary. This removes a potential ambiguity in mapping between a string of octets and an integer value.

As noted in section 4.1.2.2, serial numbers can be expected to contain long integers. Certificate users MUST be able to handle serialNumber values up to 20 octets in length. Conformant CAs MUST NOT use serialNumber values longer than 20 octets.

As noted in section 5.2.3, CRL numbers can be expected to contain long integers. CRL validators MUST be able to handle cRLNumber values up to 20 octets in length. Conformant CRL issuers MUST NOT use cRLNumber values longer than 20 octets.

The construct "SEQUENCE SIZE (1..MAX) OF" appears in several ASN.1 constructs. A valid ASN.1 sequence will have zero or more entries. The SIZE (1..MAX) construct constrains the sequence to have at least one entry. MAX indicates the upper bound is unspecified. Implementations are free to choose an upper bound that suits their environment.

The construct "positiveInt ::= INTEGER (0..MAX)" defines positiveInt as a subtype of INTEGER containing integers greater than or equal to zero. The upper bound is unspecified. Implementations are free to select an upper bound that suits their environment.

The character string type PrintableString supports a very basic Latin character set: the lower case letters ’a’ through ’z’, upper case letters ’A’ through ’Z’, the digits ’0’ through ’9’, eleven special characters ’ = ( ) + , - . / : ? and space.

Implementers should note that the at sign (’@’) and underscore (’_’) characters are not supported by the ASN.1 type PrintableString. These characters often appear in internet addresses. Such addresses MUST be encoded using an ASN.1 type that supports them. They are usually encoded as IA5String in either the emailAddress attribute within a distinguished name or the rfc822Name field of GeneralName. Conforming implementations MUST NOT encode strings which include either the at sign or underscore character as PrintableString.

The character string type TeletexString is a superset of PrintableString. TeletexString supports a fairly standard (ASCII-like) Latin character set, Latin characters with non-spacing accents and Japanese characters.

Named bit lists are BIT STRINGs where the values have been assigned names. This specification makes use of named bit lists in the definitions for the key usage, CRL distribution points and freshest CRL certificate extensions, as well as the freshest CRL and issuing distribution point CRL extensions. When DER encoding a named bit list, trailing zeroes MUST be omitted. That is, the encoded value ends with the last named bit that is set to one.

The character string type UniversalString supports any of the characters allowed by ISO 10646-1 [ISO 10646]. ISO 10646-1 is the Universal multiple-octet coded Character Set (UCS). ISO 10646-1 specifies the architecture and the "basic multilingual plane" -- a large standard character set which includes all major world character standards.

The character string type UTF8String was introduced in the 1997 version of ASN.1, and UTF8String was added to the list of choices for DirectoryString in the 2001 version of X.520 [X.520]. UTF8String is a universal type and has been assigned tag number 12. The content of UTF8String was defined by RFC 2044 [RFC 2044] and updated in RFC 2279 [RFC 2279].

In anticipation of these changes, and in conformance with IETF Best Practices codified in RFC 2277 [RFC 2277], IETF Policy on Character Sets and Languages, this document includes UTF8String as a choice in DirectoryString and the CPS qualifier extensions.

Implementers should note that the DER encoding of the SET OF values requires ordering of the encodings of the values. In particular, this issue arises with respect to distinguished names.

Implementers should note that the DER encoding of SET or SEQUENCE components whose value is the DEFAULT omit the component from the encoded certificate or CRL. For example, a BasicConstraints extension whose cA value is FALSE would omit the cA boolean from the encoded certificate.

Object Identifiers (OIDs) are used throughout this specification to identify certificate policies, public key and signature algorithms, certificate extensions, etc. There is no maximum size for OIDs. This specification mandates support for OIDs which have arc elements with values that are less than 2^28, that is, they MUST be between 0 and 268,435,455, inclusive. This allows each arc element to be represented within a single 32 bit word. Implementations MUST also support OIDs where the length of the dotted decimal (see [RFC 2252], section 4.1) string representation can be up to 100 bytes (inclusive). Implementations MUST be able to handle OIDs with up to 20 elements (inclusive). CAs SHOULD NOT issue certificates which contain OIDs that exceed these requirements. Likewise, CRL issuers SHOULD NOT issue CRLs which contain OIDs that exceed these requirements.

Implementors are warned that the X.500 standards community has developed a series of extensibility rules. These rules determine when an ASN.1 definition can be changed without assigning a new object identifier (OID). For example, at least two extension definitions included in RFC 2459 [RFC 2459], the predecessor to this profile document, have different ASN.1 definitions in this specification, but the same OID is used. If unknown elements appear within an extension, and the extension is not marked critical, those unknown elements ought to be ignored, as follows:

(a) ignore all unknown bit name assignments within a bit string;

(b) ignore all unknown named numbers in an ENUMERATED type or INTEGER type that is being used in the enumerated style, provided the number occurs as an optional element of a SET or SEQUENCE; and

(c) ignore all unknown elements in SETs, at the end of SEQUENCEs, or in CHOICEs where the CHOICE is itself an optional element of a SET or SEQUENCE.

If an extension containing unexpected values is marked critical, the implementation MUST reject the certificate or CRL containing the unrecognized extension.

Appendix C. Examples

This section contains four examples: three certificates and a CRL. The first two certificates and the CRL comprise a minimal certification path.

Section C.1 contains an annotated hex dump of a "self-signed" certificate issued by a CA whose distinguished name is cn=us,o=gov,ou=nist. The certificate contains a DSA public key with parameters, and is signed by the corresponding DSA private key.

Section C.2 contains an annotated hex dump of an end entity certificate. The end entity certificate contains a DSA public key, and is signed by the private key corresponding to the "self-signed" certificate in section C.1.

Section C.3 contains a dump of an end entity certificate which contains an RSA public key and is signed with RSA and MD5. This certificate is not part of the minimal certification path.

Section C.4 contains an annotated hex dump of a CRL. The CRL is issued by the CA whose distinguished name is cn=us,o=gov,ou=nist and the list of revoked certificates includes the end entity certificate presented in C.2.

The certificates were processed using Peter Gutman’s dumpasn1 utility to generate the output. The source for the dumpasn1 utility is available at <http://www.cs.auckland.ac.nz/˜pgut001/dumpasn1.c>. The binaries for the certificates and CRLs are available at <http://csrc.nist.gov/pki/pkixtools>.

C.1 Certificate

This section contains an annotated hex dump of a 699 byte version 3 certificate. The certificate contains the following information:
(a) the serial number is 23 (17 hex);

(b) the certificate is signed with DSA and the SHA-1 hash algorithm;
(c) the issuer’s distinguished name is OU=NIST; O=gov; C=US
(d) and the subject’s distinguished name is OU=NIST; O=gov; C=US
(e) the certificate was issued on June 30, 1997 and will expire on December 31, 1997;
(f) the certificate contains a 1024 bit DSA public key with parameters;
(g) the certificate contains a subject key identifier extension generated using method (1) of section 4.2.1.2; and
(h) the certificate is a CA certificate (as indicated through the basic constraints extension.)

```
  0 30  699: SEQUENCE {
  4 30  635:   SEQUENCE {
  8 A0    3:     [0] {
 10 02    1:       INTEGER 2
           :       }
 13 02    1:     INTEGER 17
 16 30    9:     SEQUENCE {
 18 06    7:       OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
           :       }
 27 30   42:     SEQUENCE {
 29 31   11:       SET {
 31 30    9:         SEQUENCE {
 33 06    3:           OBJECT IDENTIFIER countryName (2 5 4 6)
 38 13    2:           PrintableString ’US’
           :           }
           :         }
 42 31   12:       SET {
 44 30   10:         SEQUENCE {
 46 06    3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
 51 13    3:           PrintableString ’gov’
           :           }
           :         }
 56 31   13:       SET {
 58 30   11:         SEQUENCE {
 60 06    3:           OBJECT IDENTIFIER
           :             organizationalUnitName (2 5 4 11)
 65 13    4:           PrintableString ’NIST’
           :           }
           :         }
           :       }
 71 30   30:     SEQUENCE {
 73 17   13:       UTCTime ’970630000000Z’
 88 17   13:       UTCTime ’971231000000Z’
           :       }
103 30   42:     SEQUENCE {
105 31   11:       SET {
```

```
107 30    9:          SEQUENCE {
109 06    3:            OBJECT IDENTIFIER countryName (2 5 4 6)
114 13    2:            PrintableString 'US'
           :            }
           :          }
118 31   12:        SET {
120 30   10:          SEQUENCE {
122 06    3:            OBJECT IDENTIFIER organizationName (2 5 4 10)
127 13    3:            PrintableString 'gov'
           :            }
           :          }
132 31   13:        SET {
134 30   11:          SEQUENCE {
136 06    3:            OBJECT IDENTIFIER
           :              organizationalUnitName (2 5 4 11)
141 13    4:            PrintableString 'NIST'
           :            }
           :          }
           :        }
147 30  440:      SEQUENCE {
151 30  300:        SEQUENCE {
155 06    7:          OBJECT IDENTIFIER dsa (1 2 840 10040 4 1)
164 30  287:          SEQUENCE {
168 02  129:            INTEGER
           :              00 B6 8B 0F 94 2B 9A CE A5 25 C6 F2 ED FC
           :              FB 95 32 AC 01 12 33 B9 E0 1C AD 90 9B BC
           :              48 54 9E F3 94 77 3C 2C 71 35 55 E6 FE 4F
           :              22 CB D5 D8 3E 89 93 33 4D FC BD 4F 41 64
           :              3E A2 98 70 EC 31 B4 50 DE EB F1 98 28 0A
           :              C9 3E 44 B3 FD 22 97 96 83 D0 18 A3 E3 BD
           :              35 5B FF EE A3 21 72 6A 7B 96 DA B9 3F 1E
           :              5A 90 AF 24 D6 20 F0 0D 21 A7 D4 02 B9 1A
           :              FC AC 21 FB 9E 94 9E 4B 42 45 9E 6A B2 48
           :              63 FE 43
300 02   21:            INTEGER
           :              00 B2 0D B0 B1 01 DF 0C 66 24 FC 13 92 BA
           :              55 F7 7D 57 74 81 E5
323 02  129:            INTEGER
           :              00 9A BF 46 B1 F5 3F 44 3D C9 A5 65 FB 91
           :              C0 8E 47 F1 0A C3 01 47 C2 44 42 36 A9 92
           :              81 DE 57 C5 E0 68 86 58 00 7B 1F F9 9B 77
           :              A1 C5 10 A5 80 91 78 51 51 3C F6 FC FC CC
           :              46 C6 81 78 92 84 3D F4 93 3D 0C 38 7E 1A
           :              5B 99 4E AB 14 64 F6 0C 21 22 4E 28 08 9C
           :              92 B9 66 9F 40 E8 95 F6 D5 31 2A EF 39 A2
           :              62 C7 B2 6D 9E 58 C4 3A A8 11 81 84 6D AF
           :              F8 B4 19 B4 C2 11 AE D0 22 3B AA 20 7F EE
           :              1E 57 18
           :           }
           :         }
455 03  133:       BIT STRING 0 unused bits, encapsulates {
459 02  129:           INTEGER
           :             00 B5 9E 1F 49 04 47 D1 DB F5 3A DD CA 04
           :             75 E8 DD 75 F6 9B 8A B1 97 D6 59 69 82 D3
           :             03 4D FD 3B 36 5F 4A F2 D1 4E C1 07 F5 D1
           :             2A D3 78 77 63 56 EA 96 61 4D 42 0B 7A 1D
           :             FB AB 91 A4 CE DE EF 77 C8 E5 EF 20 AE A6
           :             28 48 AF BE 69 C3 6A A5 30 F2 C2 B9 D9 82
           :             2B 7D D9 C4 84 1F DE 0D E8 54 D7 1B 99 2E
           :             B3 D0 88 F6 D6 63 9B A7 E2 0E 82 D4 3B 8A
           :             68 1B 06 56 31 59 0B 49 EB 99 A5 D5 81 41
           :             7B C9 55
           :           }
           :       }
591 A3   50:     [3] {
593 30   48:       SEQUENCE {
595 30   29:         SEQUENCE {
597 06    3:           OBJECT IDENTIFIER
           :             subjectKeyIdentifier (2 5 29 14)
602 04   22:           OCTET STRING, encapsulates {
604 04   20:               OCTET STRING
           :                 86 CA A5 22 81 62 EF AD 0A 89 BC AD 72 41
           :                 2C 29 49 F4 86 56
           :               }
           :           }
626 30   15:         SEQUENCE {
628 06    3:           OBJECT IDENTIFIER basicConstraints (2 5 29 19)
633 01    1:           BOOLEAN TRUE
636 04    5:           OCTET STRING, encapsulates {
638 30    3:               SEQUENCE {
640 01    1:                 BOOLEAN TRUE
           :                 }
           :               }
           :           }
           :         }
           :       }
           :     }
643 30    9:   SEQUENCE {
645 06    7:     OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
           :     }
654 03   47:   BIT STRING 0 unused bits, encapsulates {
657 30   44:       SEQUENCE {
659 02   20:         INTEGER
           :           43 1B CF 29 25 45 C0 4E 52 E7 7D D6 FC B1
           :           66 4C 83 CF 2D 77
681 02   20:         INTEGER
```

```
           :           0B 5B 9A 24 11 98 E8 F3 86 90 04 F6 08 A9
           :           E1 8D A5 CC 3A D4
           :         }
           :       }
           :   }
```

C.2 Certificate

This section contains an annotated hex dump of a 730 byte version 3 certificate. The certificate contains the following information:
(a) the serial number is 18 (12 hex);
(b) the certificate is signed with DSA and the SHA-1 hash algorithm;
(c) the issuer’s distinguished name is OU=nist; O=gov; C=US
(d) and the subject’s distinguished name is CN=Tim Polk; OU=nist; O=gov; C=US
(e) the certificate was valid from July 30, 1997 through December 1, 1997;
(f) the certificate contains a 1024 bit DSA public key;
(g) the certificate is an end entity certificate, as the basic constraints extension is not present;
(h) the certificate contains an authority key identifier extension matching the subject key identifier of the certificate in Appendix C.1; and
(i) the certificate includes one alternative name - an RFC 822 address of "wpolk@nist.gov".

```
     0 30  730: SEQUENCE {
     4 30  665:   SEQUENCE {
     8 A0    3:     [0] {
    10 02    1:       INTEGER 2
              :       }
    13 02    1:     INTEGER 18
    16 30    9:     SEQUENCE {
    18 06    7:       OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
              :       }
    27 30   42:     SEQUENCE {
    29 31   11:       SET {
    31 30    9:         SEQUENCE {
    33 06    3:           OBJECT IDENTIFIER countryName (2 5 4 6)
    38 13    2:           PrintableString ’US’
              :           }
              :         }
    42 31   12:       SET {
    44 30   10:         SEQUENCE {
    46 06    3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
    51 13    3:           PrintableString ’gov’
              :           }
              :         }
    56 31   13:       SET {
    58 30   11:         SEQUENCE {
    60 06    3:           OBJECT IDENTIFIER
              :             organizationalUnitName (2 5 4 11)
    65 13    4:           PrintableString 'NIST'
              :           }
              :         }
              :       }
    71 30   30:     SEQUENCE {
    73 17   13:       UTCTime '970730000000Z'
    88 17   13:       UTCTime '971201000000Z'
              :       }
   103 30   61:     SEQUENCE {
   105 31   11:       SET {
   107 30    9:         SEQUENCE {
   109 06    3:           OBJECT IDENTIFIER countryName (2 5 4 6)
   114 13    2:           PrintableString 'US'
              :           }
              :         }
   118 31   12:       SET {
   120 30   10:         SEQUENCE {
   122 06    3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
   127 13    3:           PrintableString 'gov'
              :           }
              :         }
   132 31   13:       SET {
   134 30   11:         SEQUENCE {
   136 06    3:           OBJECT IDENTIFIER
              :             organizationalUnitName (2 5 4 11)
   141 13    4:           PrintableString 'NIST'
              :           }
              :         }
   147 31   17:       SET {
   149 30   15:         SEQUENCE {
   151 06    3:           OBJECT IDENTIFIER commonName (2 5 4 3)
   156 13    8:           PrintableString 'Tim Polk'
              :           }
              :         }
              :       }
   166 30  439:     SEQUENCE {
   170 30  300:       SEQUENCE {
   174 06    7:         OBJECT IDENTIFIER dsa (1 2 840 10040 4 1)
   183 30  287:         SEQUENCE {
   187 02  129:           INTEGER
              :             00 B6 8B 0F 94 2B 9A CE A5 25 C6 F2 ED FC
              :             FB 95 32 AC 01 12 33 B9 E0 1C AD 90 9B BC
              :             48 54 9E F3 94 77 3C 2C 71 35 55 E6 FE 4F
              :             22 CB D5 D8 3E 89 93 33 4D FC BD 4F 41 64
```

```
            :             3E A2 98 70 EC 31 B4 50 DE EB F1 98 28 0A
            :             C9 3E 44 B3 FD 22 97 96 83 D0 18 A3 E3 BD
            :             35 5B FF EE A3 21 72 6A 7B 96 DA B9 3F 1E
            :             5A 90 AF 24 D6 20 F0 0D 21 A7 D4 02 B9 1A
            :             FC AC 21 FB 9E 94 9E 4B 42 45 9E 6A B2 48
            :             63 FE 43
   319 02   21:           INTEGER
            :             00 B2 0D B0 B1 01 DF 0C 66 24 FC 13 92 BA
            :             55 F7 7D 57 74 81 E5
   342 02  129:           INTEGER
            :             00 9A BF 46 B1 F5 3F 44 3D C9 A5 65 FB 91
            :             C0 8E 47 F1 0A C3 01 47 C2 44 42 36 A9 92
            :             81 DE 57 C5 E0 68 86 58 00 7B 1F F9 9B 77
            :             A1 C5 10 A5 80 91 78 51 51 3C F6 FC FC CC
            :             46 C6 81 78 92 84 3D F4 93 3D 0C 38 7E 1A
            :             5B 99 4E AB 14 64 F6 0C 21 22 4E 28 08 9C
            :             92 B9 66 9F 40 E8 95 F6 D5 31 2A EF 39 A2
            :             62 C7 B2 6D 9E 58 C4 3A A8 11 81 84 6D AF
            :             F8 B4 19 B4 C2 11 AE D0 22 3B AA 20 7F EE
            :             1E 57 18
            :           }
            :         }
   474 03  132:       BIT STRING 0 unused bits, encapsulates {
   478 02  128:           INTEGER
            :             30 B6 75 F7 7C 20 31 AE 38 BB 7E 0D 2B AB
            :             A0 9C 4B DF 20 D5 24 13 3C CD 98 E5 5F 6C
            :             B7 C1 BA 4A BA A9 95 80 53 F0 0D 72 DC 33
            :             37 F4 01 0B F5 04 1F 9D 2E 1F 62 D8 84 3A
            :             9B 25 09 5A 2D C8 46 8E 2B D4 F5 0D 3B C7
            :             2D C6 6C B9 98 C1 25 3A 44 4E 8E CA 95 61
            :             35 7C CE 15 31 5C 23 13 1E A2 05 D1 7A 24
            :             1C CB D3 72 09 90 FF 9B 9D 28 C0 A1 0A EC
            :             46 9F 0D B8 D0 DC D0 18 A6 2B 5E F9 8F B5
            :             95 BE
            :           }
            :       }
   609 A3   62:     [3] {
   611 30   60:       SEQUENCE {
   613 30   25:         SEQUENCE {
   615 06    3:           OBJECT IDENTIFIER subjectAltName (2 5 29 17)
   620 04   18:           OCTET STRING, encapsulates {
   622 30   16:               SEQUENCE {
   624 81   14:                 [1] 'wpolk@nist.gov'
            :                 }
            :               }
            :           }
   640 30   31:         SEQUENCE {
   642 06    3:           OBJECT IDENTIFIER
```

```
             :             authorityKeyIdentifier (2 5 29 35)
   647 04   24:           OCTET STRING, encapsulates {
   649 30   22:               SEQUENCE {
   651 80   20:                 [0]
              :                   86 CA A5 22 81 62 EF AD 0A 89 BC AD 72
              :                   41 2C 29 49 F4 86 56
              :                 }
              :               }
              :           }
              :         }
              :       }
              :     }
   673 30    9:   SEQUENCE {
   675 06    7:     OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
              :     }
   684 03   48:   BIT STRING 0 unused bits, encapsulates {
   687 30   45:       SEQUENCE {
   689 02   20:         INTEGER
              :           36 97 CB E3 B4 2C E1 BB 61 A9 D3 CC 24 CC
              :           22 92 9F F4 F5 87
   711 02   21:         INTEGER
              :           00 AB C9 79 AF D2 16 1C A9 E3 68 A9 14 10
              :           B4 A0 2E FF 22 5A 73
              :         }
              :       }
              :   }
```

C.3 End Entity Certificate Using RSA

This section contains an annotated hex dump of a 654 byte version 3 certificate. The certificate contains the following information:
(a) the serial number is 256;
(b) the certificate is signed with RSA and the SHA-1 hash algorithm;
(c) the issuer’s distinguished name is OU=NIST; O=gov; C=US
(d) and the subject’s distinguished name is CN=Tim Polk; OU=NIST; O=gov; C=US
(e) the certificate was issued on May 21, 1996 at 09:58:26 and expired on May 21, 1997 at 09:58:26;
(f) the certificate contains a 1024 bit RSA public key;
(g) the certificate is an end entity certificate (not a CA certificate);
(h) the certificate includes an alternative subject name of "<http://www.itl.nist.gov/div893/staff/polk/index.html>" and an alternative issuer name of "<http://www.nist.gov/>" - both are URLs;
(i) the certificate include an authority key identifier extension and a certificate policies extension specifying the policy OID 2.16.840.1.101.3.2.1.48.9; and

(j) the certificate includes a critical key usage extension specifying that the public key is intended for verification of digital signatures.

```
  0 30  654: SEQUENCE {
  4 30  503:   SEQUENCE {
  8 A0    3:     [0] {
 10 02    1:       INTEGER 2
           :       }
 13 02    2:     INTEGER 256
 17 30   13:     SEQUENCE {
 19 06    9:       OBJECT IDENTIFIER
           :         sha1withRSAEncryption (1 2 840 113549 1 1 5)
 30 05    0:       NULL
           :       }
 32 30   42:     SEQUENCE {
 34 31   11:       SET {
 36 30    9:         SEQUENCE {
 38 06    3:           OBJECT IDENTIFIER countryName (2 5 4 6)
 43 13    2:           PrintableString ’US’
           :           }
           :         }
 47 31   12:       SET {
 49 30   10:         SEQUENCE {
 51 06    3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
 56 13    3:           PrintableString ’gov’
           :           }
           :         }
 61 31   13:       SET {
 63 30   11:         SEQUENCE {
 65 06    3:           OBJECT IDENTIFIER
           :             organizationalUnitName (2 5 4 11)
 70 13    4:           PrintableString ’NIST’
           :           }
           :         }
           :       }
 76 30   30:     SEQUENCE {
 78 17   13:       UTCTime ’960521095826Z’
 93 17   13:       UTCTime ’970521095826Z’
           :       }
108 30   61:     SEQUENCE {
110 31   11:       SET {
112 30    9:         SEQUENCE {
114 06    3:           OBJECT IDENTIFIER countryName (2 5 4 6)
119 13    2:           PrintableString ’US’
           :           }
           :         }
123 31   12:       SET {
```

```
125 30   10:          SEQUENCE {
127 06    3:            OBJECT IDENTIFIER organizationName (2 5 4 10)
132 13    3:            PrintableString 'gov'
           :            }
           :          }
137 31   13:        SET {
139 30   11:          SEQUENCE {
141 06    3:            OBJECT IDENTIFIER
           :              organizationalUnitName (2 5 4 11)
146 13    4:            PrintableString 'NIST'
           :            }
           :          }
152 31   17:        SET {
154 30   15:          SEQUENCE {
156 06    3:            OBJECT IDENTIFIER commonName (2 5 4 3)
161 13    8:            PrintableString 'Tim Polk'
           :            }
           :          }
           :        }
171 30  159:      SEQUENCE {
174 30   13:        SEQUENCE {
176 06    9:          OBJECT IDENTIFIER
           :            rsaEncryption (1 2 840 113549 1 1 1)
187 05    0:          NULL
           :          }
189 03  141:        BIT STRING 0 unused bits, encapsulates {
193 30  137:            SEQUENCE {
196 02  129:              INTEGER
           :                00 E1 6A E4 03 30 97 02 3C F4 10 F3 B5 1E
           :                4D 7F 14 7B F6 F5 D0 78 E9 A4 8A F0 A3 75
           :                EC ED B6 56 96 7F 88 99 85 9A F2 3E 68 77
           :                87 EB 9E D1 9F C0 B4 17 DC AB 89 23 A4 1D
           :                7E 16 23 4C 4F A8 4D F5 31 B8 7C AA E3 1A
           :                49 09 F4 4B 26 DB 27 67 30 82 12 01 4A E9
           :                1A B6 C1 0C 53 8B 6C FC 2F 7A 43 EC 33 36
           :                7E 32 B2 7B D5 AA CF 01 14 C6 12 EC 13 F2
           :                2D 14 7A 8B 21 58 14 13 4C 46 A3 9A F2 16
           :                95 FF 23
328 02    3:              INTEGER 65537
           :              }
           :            }
           :        }
333 A3  175:      [3] {
336 30  172:        SEQUENCE {
339 30   63:          SEQUENCE {
341 06    3:            OBJECT IDENTIFIER subjectAltName (2 5 29 17)
346 04   56:            OCTET STRING, encapsulates {
348 30   54:                SEQUENCE {
```

```
350 86   52:                 [6]
           :                   'http://www.itl.nist.gov/div893/staff/'
           :                   'polk/index.html'
           :                 }
           :               }
           :           }
404 30   31:         SEQUENCE {
406 06    3:           OBJECT IDENTIFIER issuerAltName (2 5 29 18)
411 04   24:           OCTET STRING, encapsulates {
413 30   22:               SEQUENCE {
415 86   20:                 [6] 'http://www.nist.gov/'
           :                 }
           :               }
           :           }
437 30   31:         SEQUENCE {
439 06    3:           OBJECT IDENTIFIER
           :             authorityKeyIdentifier (2 5 29 35)
444 04   24:           OCTET STRING, encapsulates {
446 30   22:               SEQUENCE {
448 80   20:                 [0]
           :                   08 68 AF 85 33 C8 39 4A 7A F8 82 93 8E
           :                   70 6A 4A 20 84 2C 32
           :                 }
           :               }
           :           }
470 30   23:         SEQUENCE {
472 06    3:           OBJECT IDENTIFIER
           :             certificatePolicies (2 5 29 32)
477 04   16:           OCTET STRING, encapsulates {
479 30   14:               SEQUENCE {
481 30   12:                 SEQUENCE {
483 06   10:                   OBJECT IDENTIFIER
           :                            '2 16 840 1 101 3 2 1 48 9'
           :                   }
           :                 }
           :               }
           :           }
495 30   14:         SEQUENCE {
497 06    3:           OBJECT IDENTIFIER keyUsage (2 5 29 15)
502 01    1:           BOOLEAN TRUE
505 04    4:           OCTET STRING, encapsulates {
507 03    2:               BIT STRING 7 unused bits
           :                 '1'B (bit 0)
           :               }
           :           }
           :         }
           :       }
           :     }
```

```
511 30   13:   SEQUENCE {
513 06    9:     OBJECT IDENTIFIER
           :       sha1withRSAEncryption (1 2 840 113549 1 1 5)
524 05    0:     NULL
           :     }
526 03  129:   BIT STRING 0 unused bits
           :     1E 07 77 6E 66 B5 B6 B8 57 F0 03 DC 6F 77
           :     6D AF 55 1D 74 E5 CE 36 81 FC 4B C5 F4 47
           :     82 C4 0A 25 AA 8D D6 7D 3A 89 AB 44 34 39
           :     F6 BD 61 1A 78 85 7A B8 1E 92 A2 22 2F CE
           :     07 1A 08 8E F1 46 03 59 36 4A CB 60 E6 03
           :     40 01 5B 2A 44 D6 E4 7F EB 43 5E 74 0A E6
           :     E4 F9 3E E1 44 BE 1F E7 5F 5B 2C 41 8D 08
           :     BD 26 FE 6A A6 C3 2F B2 3B 41 12 6B C1 06
           :     8A B8 4C 91 59 EB 2F 38 20 2A 67 74 20 0B
           :     77 F3
           :   }
```

C.4 Certificate Revocation List

This section contains an annotated hex dump of a version 2 CRL with one extension (cRLNumber). The CRL was issued by OU=NIST; O=gov; C=US on August 7, 1997; the next scheduled issuance was September 7, 1997. The CRL includes one revoked certificates: serial number 18 (12 hex), which was revoked on July 31, 1997 due to keyCompromise. The CRL itself is number 18, and it was signed with DSA and SHA-1.

```
  0 30  203: SEQUENCE {
  3 30  140:   SEQUENCE {
  6 02    1:     INTEGER 1
  9 30    9:     SEQUENCE {
 11 06    7:       OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
           :       }
 20 30   42:     SEQUENCE {
 22 31   11:       SET {
 24 30    9:         SEQUENCE {
 26 06    3:           OBJECT IDENTIFIER countryName (2 5 4 6)
 31 13    2:           PrintableString 'US'
           :           }
           :         }
 35 31   12:       SET {
 37 30   10:         SEQUENCE {
 39 06    3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
 44 13    3:           PrintableString 'gov'
           :           }
           :         }
 49 31   13:       SET {
 51 30   11:         SEQUENCE {
```

```
 53 06    3:           OBJECT IDENTIFIER
           :             organizationalUnitName (2 5 4 11)
 58 13    4:           PrintableString ’NIST’
           :           }
           :         }
           :       }
 64 17   13:     UTCTime ’970807000000Z’
 79 17   13:     UTCTime ’970907000000Z’
 94 30   34:     SEQUENCE {
 96 30   32:       SEQUENCE {
 98 02    1:         INTEGER 18
101 17   13:         UTCTime ’970731000000Z’
116 30   12:         SEQUENCE {
118 30   10:           SEQUENCE {
120 06    3:             OBJECT IDENTIFIER cRLReason (2 5 29 21)
125 04    3:             OCTET STRING, encapsulates {
127 0A    1:                 ENUMERATED 1
           :                 }
           :             }
           :           }
           :         }
           :       }
130 A0   14:     [0] {
132 30   12:       SEQUENCE {
134 30   10:         SEQUENCE {
136 06    3:           OBJECT IDENTIFIER cRLNumber (2 5 29 20)
141 04    3:           OCTET STRING, encapsulates {
143 02    1:               INTEGER 12
           :               }
           :           }
           :         }
           :       }
           :     }
146 30    9:   SEQUENCE {
148 06    7:     OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
           :     }
157 03   47:   BIT STRING 0 unused bits, encapsulates {
160 30   44:       SEQUENCE {
162 02   20:         INTEGER
           :           22 4E 9F 43 BA 95 06 34 F2 BB 5E 65 DB A6
           :           80 05 C0 3A 29 47
184 02   20:         INTEGER
           :           59 1A 57 C9 82 D7 02 21 14 C3 D4 0B 32 1B
           :           96 16 B1 1F 46 5A
           :         }
           :       }
           :   }
```

Author Addresses

Russell Housley
RSA Laboratories
918 Spring Knoll Drive
Herndon, VA 20170
USA

EMail: rhousley@rsasecurity.com

Warwick Ford
VeriSign, Inc.
401 Edgewater Place
Wakefield, MA 01880
USA

EMail: wford@verisign.com

Tim Polk
NIST
Building 820, Room 426
Gaithersburg, MD 20899
USA

EMail: wpolk@nist.gov

David Solo
Citigroup
909 Third Ave, 16th Floor
New York, NY 10043
USA

EMail: dsolo@alum.mit.edu

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.