PKIX Working Group
Internet Draft

R. Housley (SPYRUS)
W. Ford (Verisign)
W. Polk (NIST)
D. Solo (BBN)

expires in six months

December 1996

Internet Public Key Infrastructure

Part I: X.509 Certificate and CRL Profile

<draft-ietf-pkix-ipki-part1-03.txt>

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet- Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Abstract

This is the second draft of the Internet Public Key Infrastructure X.509 Certificate and CRL Profile. Since the first version was distributed, ISO has completed work on X.509 Version 3 Certificates and X.509 Version 2 Certificate Revocation Lists (CRLs). Many of the Internet community requirements that were in the previous version of this document have been included in the final ISO document. As a result, this document has gotten simpler. Please send comments on this document to the ietf-pkix@tandem.com mail list.

1 Executive Summary

This specification is Part 1 of a four part standard for development of a Public Key Infrastructure for the Internet. This specification is a standalone document; implementations of this standard may proceed before completion of parts two through four.

This specification profiles the format and semantics of certificates and certificate revocation lists for the Internet PKI. Procedures are described for processing of certification paths in the Internet environment. Encoding rules are provided for popular cryptographic algorithms. Finally, a comprehensive ASN.1 module is provided in the appendices for all data structure defined or referenced.

The specification presents profiles of the X.509 version 3 certificate and version 2 certificate revocation lists. The profiles include the identification of ISO and ANSI extensions which may be useful in the Internet PKI and definition of new extensions to meet the Internet's requirements. The profiles are presented in the 1988 Abstract Syntax Notation One (ASN.1) rather than the 1993 syntax used in the ISO standards.

This specification also includes path validation procedures. These procedures are based upon the ISO definition, but incorporate the Internet defined extensions. Implementations are required to derive the same results but are not required to use the specified procedures.

Finally, the specification describes procedures for identification and encoding of public key materials and digital signatures. Implementations are not required to use any particular cryptographic algorithms. However, conforming implementations which use the identified algorithms are required to identify and encode the public key materials and digital signatures as described.

An Appendix is provided containing all ASN.1 structures defined or referenced within this specification. As above, the material is presented in the 1988 Abstract Syntax Notation One (ASN.1) rather than the 1993 syntax.

2 Requirements and Assumptions

Goal is to develop a profile and associated management structure to facilitate the adoption/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, as well as others. 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, as opposed to syntax.

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).

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 be more appropriate method 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 availablity. 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. The profile does not prohibit the use of an X.500 Directory, but other means of distributing certificates and certificate revocation lists (CRLs) are supported.

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 sophistication/attentiveness of the users themselves. This manifests itself in minimal user configuration responsibility (e.g., root 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 users, the Internet PKI profile is structured to support the individuals who generally operate Certification Authorities (CAs). Providing administrators with unbounded choices increases the chances that a subtle CA administrator mistake will result in broad compromise. Also, unbounded choices greatly complicates the software that must 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.

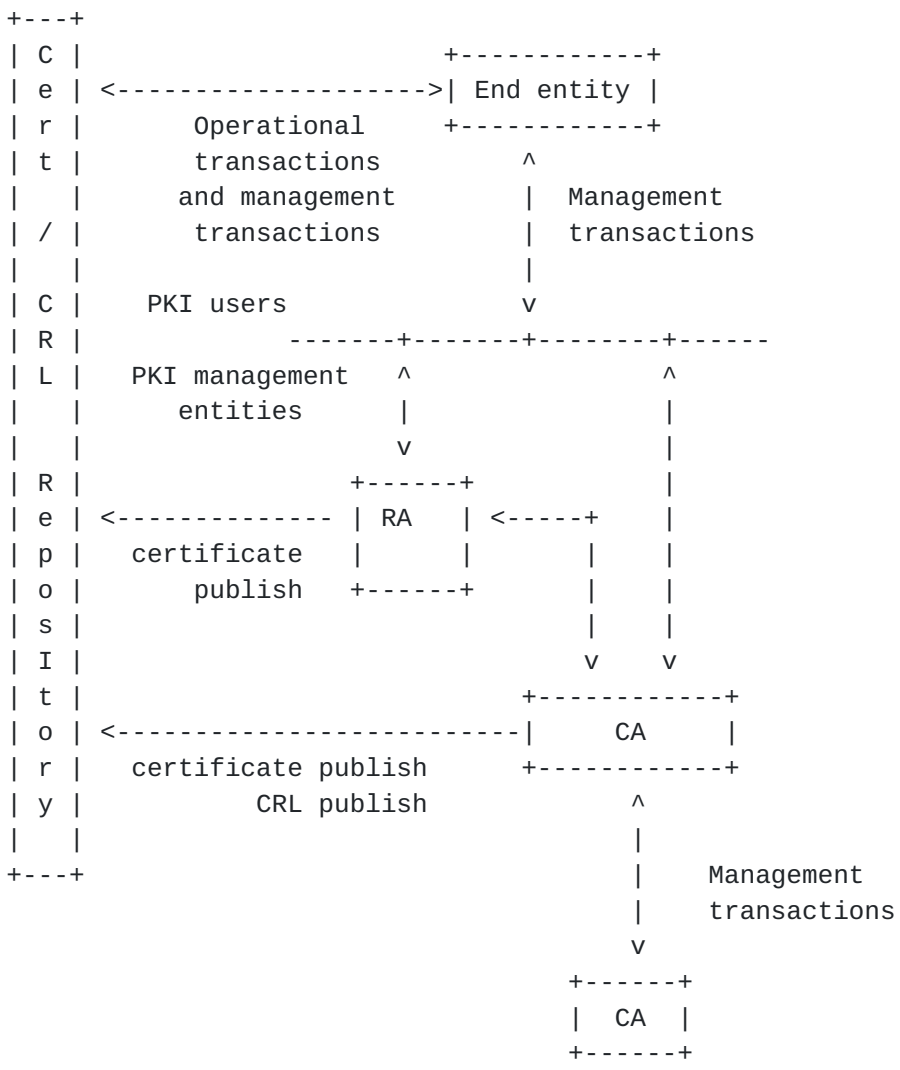

Figure 1 - PKI Entities

The components in this model are:

end entity: user of PKI certificates and/or end user system that the PKI certifies;
CA: certification authority;
RA: registration authority, i.e., an optional system to which a CA delegates certain management functions;
repository: a system or collection of distributed systems that store certificates and CRLs and serves as a means of distributing these certificates and CRLs to end entities.

3.1 X.509 Version 3 Certificate

Application of public key technology requires the user of a public key to be confident that the public key belongs to 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 subject identities. The binding is achieved by having a trusted certification authority (CA) digitally sign each certificate. 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.

The standard known as 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. 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. These two fields are used to support directory access control.

The Internet Privacy Enhanced Mail (PEM) proposals, 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 has proven necessary. In response to these new requirements, ISO/IEC 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-AM].

ISO/IEC and ANSI X9 have also developed a set of standard extensions for use in the v3 extensions field [X.509-AM][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 and ANSI 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, 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 (often one) 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 must 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 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 CA hierarchical model has been found to have several deficiencies, including:

(a) The pure top-down hierarchy, with all certification paths starting from the root, is too restrictive for many purposes. For some applications, verification of certification paths should start with a public key of a CA in a user's own domain, rather than mandating that verification commence at the top of a hierarchy. In many environments, the local domain is often the most trusted. Also, initialization and key-pair-update operations can be more effectively conducted between an end entity and a local management system.

(b) The name subordination rule introduces undesirable constraints upon the X.500 naming system an organization may use.

(c) Use of the PCA concept requires knowledge of individual PCAs to be built into certificate chain verification logic. In the particular case of Internet mail, this is not a major problem -- the PCA name can always be displayed to the human user who can make a decision as to what trust to imply from a particular chain. However, in many commercial applications, such as electronic commerce or EDI, operator intervention to make policy decisions is impractical. The process needs to be automated to a much higher degree. In fact, the full process of certificate chain processing needs to be implementable in trusted software.

Because of the above shortcomings, it is proposed that more flexible CA structures than the RFC 1422 hierarchy be supported by the PKIX specifications. In fact, the main reason for the structural restrictions imposed by RFC 1422 was the restricted certificate format provided with X.509 v1. 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.

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 might 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 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 CA issues a new CRL on a regular periodic basis (e.g., hourly, daily, or weekly). Entries are added to CRLs as revocations occur, and an entry may be removed when the certificate expiration date is reached.

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

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 the next periodic CRL is issued -- this may be up to one hour, one day, or one week depending on the frequency that the CA issues CRLs.

Another potential problem with CRLs is the risk of a CRL growing to an entirely unacceptable size. In the 1988 and 1993 versions of X.509, the CRL for the end-user certificates needed to cover the entire population of end-users for one CA. It is desirable to allow such populations to be in the range of thousands, tens of thousands, or possibly even hundreds of thousands of users. The end-user CRL is therefore at risk of growing to such sizes, which present major communication and storage overhead problems. With the version 2 CRL format, introduced along with the v3 certificate format, it becomes

possible to arbitrarily divide the population of certificates for one CA into a number of partitions, each partition being associated with one CRL distribution point (e.g., directory entry or URL) from which CRLs are distributed. Therefore, the maximum CRL size can be controlled by a CA. Separate CRL distribution points can also exist for different revocation reasons. For example, routine revocations (e.g., name change) may be placed on a different CRL to revocations resulting from suspected key compromises, and policy may specify that the latter CRL be updated and issued more frequently than the former.

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 such profiles.

Furthermore, it is recognized that on-line methods of revocation notification may be applicable in some environments as an alternative to the X.509 CRL. On-line revocation checking eliminates the latency between a revocation report and CRL the next issue. Once the revocation is reported, any query to the on-line service will correctly reflect the certificate validation impacts of the revocation. Therefore, this profile will also consider standard approaches to on-line revocation notification.

3.4 Operational Protocols

Operational protocols are required to deliver certificates and CRLs to certificate using client systems. Provision is needed for a variety of different means of certificate and CRL delivery, including request/delivery procedures based on E-mail, http, X.500, and WHOIS++. These specifications include definitions of, and/or references to, 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 Public Key Infrastructure (PKI) components. For example, 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, 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 in it necessary 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 of a CA, 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 public 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 associated with a CA. 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 the information necessary to establish cross-certificates between those CAs.

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 through 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.

Part 3 of 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 (on-line, e-mail, and WWW) are also specified.

4 Certificate and Certificate Extensions Profile

This section presents a profile for public key certificates that will foster interoperability and a reusable public key infrastructure.

This section is based upon the X.509 V3 certificate format and the standard certificate extensions defined in the Amendment [X.509-AM]. The ISO definitions use the 1993 version of ASN.1; while this document uses the older ASN.1 syntax, the encoded certificate and standard extensions are equivalent. This section also defines private extensions required to support a public key infrastructure 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. Other efforts are looking at certificate profiles for payment systems.

4.1 Basic Certificate Fields

The X.509 v3 certificate basic syntax is as follows. For signature calculation, the certificate is encoded using the ASN.1 distinguished encoding rules (DER) [X.208]. ASN.1 DER encoding is a tag, length, value encoding system for each element.

```
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      CertificateValidityDate,
     notAfter       CertificateValidityDate }

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

UniqueIdentifier  ::=  BIT STRING

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

Extensions  ::=  SEQUENCE OF Extension

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

The following items describe a proposed use of the X.509 v3 certificate for the Internet.

4.1.1 Certificate Fields

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

4.1.1.1 tbsCertificate

The first field in the sequence is the tbsCertificate. This is a itself a sequence, and contains the names of the subject and issuer, a public key associated with the subject an expiration date, and other associated information. The fields of the basic tbsCertificate are described in detail in section 4.1.2; the tbscertificate may also include extensions which are described in section 4.2.

4.1.1.2 signatureAlgorithm

The signatureAlgorithm field contains the algorithm identifier for the algorithm used by the CA to sign the certificate. Section 7.2 lists the supported signature algorithms.

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

4.1.1.3 signature

The signature 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 a one-way hash function. The one-way hash function output value is ASN.1 encoded as an OCTET STRING and the result is encrypted (e.g., using RSA Encryption) to form the signed quantity. This signature value is then ASN.1 encoded as a BIT STRING and included in the Certificate's signature field.

By generating this signature, a CA certifies the validity of the information in tbscertificate. 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 is a sequence which 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, an expiration date, a version number and a serial number; some will contain optional unique identifier fields. The remainder of this section describes the syntax and semantics of these fields. A TBSCertificate may also include 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, use X.509 version 3 (value is 2). If no extensions are present, but a UniqueIdentifier is present, use version 2 (value is 1). If only basic fields are present, use version 1 (the value is omitted from the certificate as the default value).

Implementations should be prepared to accept any version certificate. In particular, at a minimum, implementations must recognize version 3 certificates; determine whether any critical extensions are present; and accept certificates without critical extensions even if they don't recognize any extensions. A certificate with an unrecognized critical extension must always be rejected.

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

4.1.2.2 Serial number

The serial number is an integer assigned by the certification authority 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).

4.1.2.3 Signature

This field contains the algorithm identifier for the algorithm used by the CA to sign the certificate. Section 7.2 lists the supported signature algorithms.

4.1.2.4 Issuer Name

The issuer name identifies the entity who has signed (and issued the certificate). The issuer identity may be carried in the issuer name field and/or the issuerAltName extension. If identity information is present only in the issuerAltName extension, then the issuer name may be an empty sequence and the issuerAltName extension must be critical.

4.1.2.5 Validity

This field indicates the dates on which the certificate becomes valid (notBefore) and on which the certificate ceases to be valid (notAfter). Both notBefore and notAfter may be encoded as UTCTime or GeneralizedTime.

CAs conforming to this profile shall not issue certificates where notAfter or notBefore is encoded as GeneralizedTime before the year 2005. CAs conforming to this profile shall not issue certificates where notAfter or notBefore is encoded as UTCTime after the year 2015.

4.1.2.5.1 UTCTime

The universal time type, UTCTime, is a standard ASN.1 type intended for international applications where local time alone is not adequate. 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 shall be expressed Greenwich Mean Time (Zulu) and shall include< seconds (i.e., times are YYMMDDHHMMSSZ), even where the number of seconds is zero. Conforming systems shall interpret the year field (YY) as follows:

Where YY is greater than 50, the year shall be interpreted as 19YY; and

Where YY is less than or equal to 50, the year shall be interpreted as 20YY.

4.1.2.6 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 shall be expressed Greenwich Mean Time (Zulu) and shall include seconds (i.e., times are YYYYMMDDHHMMSSZ), even where the number of seconds is zero. GeneralizedTime values shall not include fractional seconds.

4.1.2.6 Subject Name

The subject name identifies the entity associated with the public key stored in the subject public key field. The subject identity may be carried in the subject field and/or the subjectAltName extension. If identity information is present only in the subjectAltName extension (e.g., a key bound only to an email address or URI), then the subject name may be an empty sequence and the subjectAltName extension must be critical.

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.

4.1.2.8 Unique Identifiers

The subject and issuer unique identifier 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 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 and making comparisons.

4.2 Certificate Extensions

The extensions defined for X.509 v3 certificates provide methods for associating additional attributes with users or public keys, for

managing the certification hierarchy, and for managing CRL distribution. 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 may be designated as critical or non-critical. A certificate using system (an application validating a certificate) must reject the certificate if it encounters a critical extension it does not recognize. A non-critical extension may be ignored if it is not recognized. The following presents recommended extensions used within Internet certificates and standard locations for information. Communities may elect to use additional extensions; however, caution should be exercised in adopting any critical extensions in certificates which might be used in a general context.

Each extension includes an object identifier and an ASN.1 structure. When an extension appears in a certificate, the object identifier appears as the field extnID and the corresponding ASN.1 encoded structure is the value of the bit string extnValue. Only one instance of a particular extension may appear in a particular certificate. For example, a certificate may contain only one authority key identifier extension (4.2.1.1). An extension may also include the optional boolean critical; critical's default value is FALSE. The text for each extension specifies the acceptable values for the critical field.

Conforming CAs are required to support the Basic Constraints extension (Section 4.2.1.10). If the CA issues certificates with an empty sequence for the subject field, the CA must support the altSubjectName extension. If the CA issues certificates with an empty sequence for the issuer field, the CA must support the altIssuerName extension. 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 shall recognize extensions which shall or may be critical. These extensions are: key usage (4.2.1.3), certificate policies (4.2.1.5), the alternative subject name (4.2.1.7), issuer alternative name (4.2.1.8), basic constraints (4.2.1.10), name constraints (4.2.1.11), policy constraints (4.2.1.12), authority information access (4.2.2.2), and CA information access (4.2.2.3).

In addition, this profile recommends support for key identifiers (4.2.1.2 and 4.2.1.3) and CRL distribution points (4.2.1.13).

4.2.1 Standard Extensions

This section identifies standard certificate extensions defined in [X.509-AM] for use in the Internet Public Key Infrastructure. Each extension is associated with an object identifier defined in [X.509-AM]. These object identifiers are members of the certificateExtension arc, which is defined by the following:

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

4.2.1.1 Authority Key Identifier

The authority key identifier extension provides a means of identifying the particular public key used to sign a certificate. This extension would be used where an issuer has multiple signing keys (either due to multiple concurrent key pairs or due to changeover). In general, this extension should be included in certificates.

The identification can 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 key identifier method is recommended in this profile. Conforming CAs that generate this extension shall include or omit both authorityCertIssuer and authorityCertSerialNumber. If authorityCertIssuer and authorityCertSerialNumber are omitted, the keyIdentifier field shall be present.

This extension shall 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 the particular public key used in an application. Where a reference to a public key identifier is needed (as with an Authority Key

Identifier) and one is not included in the associated certificate, a SHA-1 hash of the subject public key shall be used. The hash shall

be calculated over the value (excluding tag and length) of the subject public key field in the certificate. This extension should be marked non-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 multipurpose key is to be restricted (e.g., when an RSA key should be used only for signing or only for key encipherment). The profile recommends that when used, this be marked as a critical extension.

```
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) }
```

4.2.1.4 Private Key Usage Period

The private key usage period extension allows the certificate issuer to specify a different validty 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 shall not generate certificates with private key usage period extensions unless at least one of the two components is present.

This profile recommends against the use of this extension. CAs conforming to this profile shall not generate certificates with critical private key usage period extensions.

```
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 policy information terms, each of which consists of an object identifier (OID) and optional qualifiers. These policy information terms indicate the policy under which the certificate has been issued and the purposes for which the certificate may be used. This profile strongly recommends that a simple OID be present in this field. Optional qualifiers which may be present are expected to provide information about obtaining CA rules, not change the definition of the policy.

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, or must reject the certificate. (Applications are free to ignore the policy field, even if the extension is marked critical.)

This specification defines two policy qualifiers types for use by certificate policy writers and certificate issuers at their own discretion. The quailfier types are the CPS Pointer qualifier, and the User Notice qualifier.

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.

The User Notice qualifier contains a text string that is to be displayed to a certificate user (including subscribers and relying parties) prior to the use of the certificate. The text string may be an IA5String or a BMPString - a subset of the ISO 100646-1 multiple octet coded character set. A CA may invoke a procedure that requires that the certficate user acknowledge that the applicable terms and conditions have been disclosed or accepted.

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

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-pkix-cps      OBJECT IDENTIFIER ::=  { pkix 4 }
id-pkix-unotice  OBJECT IDENTIFIER ::=  { pkix 5 }

PolicyQualifierId ::= ENUMERATED { id-pkix-cps, id-pkix-unotice }

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

CPSuri ::= IA5String

UserNotice ::= CHOICE {
  ia5String     IA5String,
  bmpString     ANY }
```

4.2.1.6 Policy Mappings

This extension is used in CA certificates. It lists pairs of obbjectidentifiers; 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 may accept an issuerDomainPolicy for certain applications. The policy mapping tells the issuing CA's users which policies associated with the subject CA are comparable to the policy they accept.

This extension may be supported by CAs and/or applications, and it is always 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 rfc822 name (electronic mail address), a DNS name, an IP address, and a URI. Other options exist, including completely local definitions. Multiple instances of a name and multiple name forms may be included. Whenever such identities are to be bound into a certificate, the subject alternative name (or issuer alternative name) extension shall be used. (Note: a form of such an identifier may also be present in the subject distinguished name; however, the alternative name extension is the preferred location for finding such information.)

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 should be empty (an empty sequence), the subjectAltName extension should be used. If the subject field contains an empty squence, the subjectAltName extension shall be marked critical.

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.

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

SubjectAltName ::= GeneralNames

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

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

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

4.2.1.8 Issuer Alternative Name

As with 4.2.1.7, this extension is used to associate Internet style identities with the certificate issuer. If the only issuer identity included in the certificate is an alternative name form (e.g., an electronic mail address), then the issuer distinguished name should be empty (an empty sequence), the issuerAltName extension should be used. If the issuer field is empty and more than one issuerAltName extension is included in the certificate, the issuerAltName extension shall 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 not recommended as an essential part of this profile, but it may
be used in local environments. This extension is always 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 how deep a certification path may exist through that CA. This profile requires the use of this extension, and it shall be critical for all certificates issued to CAs.

```
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 provides permitted and excluded subtrees that place restrictions on names that may be included within a certificate issued by a given CA. Restrictions may apply to the subject distringuished name or subject alternative names. Any name matching a restriction in the excluded subtrees field is invalid regardless of information appearing in the permitted subtrees. This extension may be critical or non-critical.

Restrictions for the rfc822, dNSName, and uri name forms are all expressed in terms of strings with wild card matching. An "*" is the wildcard character. The minimum and maximum fields in general subtree are not used for these name forms. For uris and rfc822 names, the restriction applies to the host part of the name. Examples would be foo.bar.com; www*.bar.com; *.xyz.com.

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

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 limit the set of poicies that can in subsequent certificates. This extension may be critical or non-critical.

```
id-ce-policyConstraints OBJECT IDENTIFIER ::=  { id-ce 34 }

PolicyConstraints ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
     policySet                     [0] CertPolicySet OPTIONAL,
     requireExplicitPolicy         [1] SkipCerts OPTIONAL,
     inhibitPolicyMapping          [2] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)

CertPolicySet ::= SEQUENCE SIZE (1..MAX) OF CertPolicyId
```

4.2.1.13 CRL Distribution Points

The CRL distribution points extension identifies how CRL information is obtained. The extension shall 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.

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

cRLDistributionPoints ::= {
     CRLDistPointsSyntax }

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

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

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

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

4.2.2 Private Internet Extensions

This section defines new extensions for use in the Internet Public Key Infrastructure. These extensions may be used to direct applications to additional information about the certificate's subject or issuer. This extension includes the type of information, where it is located, and a method for obtaining the information. The types of information include certificate status, CA policy information, and related certificates.

The object identifiers associated with the private extensions are defined under the iso (1), org (3), dod (6), internet (1), security(5) or 1.3.6.1.5, branch of the name space. These extensions make use of OIDs of the form {applTCPProtoID port}, which identify TCP-based protocols that don't have OIDs assigned by other means, to identify common methods for retrieving information.

The following ASN.1 defines object identifiers which may be used by applications that implement the private extensions; additional access methods may be used, but the semantics are undefined by this document.

```
pkix  OBJECT IDENTIFIER ::= { iso(1) org(3) dod (6) internet (1)
               security(5) pkix(?) }

-- Object identifiers for ftp, http, smtp and ldap protocols

applTCPProto OBJECT IDENTIFIER ::= { 1 3 6 1 2 1 27 4 }

ftpID    OBJECT-IDENTIFIER ::= {applTCPProtoID 21}
httpID   OBJECT-IDENTIFIER ::= {applTCPProtoID 80}
smtpID   OBJECT-IDENTIFIER ::= {applTCPProtoID 25}
ldapID   OBJECT-IDENTIFIER ::= {applTCPProtoID 389}

-- Object identifier for the X.500 directory access protocol

dap OBJECT-IDENTIFIER ::= { 2 5 3 1 }
```

4.2.2.1 Subject Information Access

The name information in the certificate identifies the entity to which the public key is bound. In some instances, it may also be necessary to know where to find additional information about the named entity. In the case of X.500 names, this relationship is automatic. The subject information access extension provides a means of identifying where and how to find information about the subject. The extension specifies a method of obtaining information and a general name form indicating where. This extension shall always be non-critical.

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

-- subjectInfoAccess ::=  { SubjectInfoAccessSyntax }

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

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

This specification defines the following values for accessMethod: ftpID, httpID, smtpID, ldapID, and dap. The accessMethod value indicates the protocol required to obtain the information. If accessMethod is ftpID, then the information must available through anonymous ftp. If accessMethod is ftpID, httpID, smtpID, or ldapID,

then the accessLocation shall be a uniformResourceIndicator (i.e., a URI). The URI shall specify all information required to retrieve the information. If accessMethod is dap, then the accessLocation shall be a directoryName.

4.2.2.2 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 include certificate status or on-line validation services, certificate retrieval, CA policy data, and CA certificates (certificates certifying the target CA to aid in certification path navigation). This extension may be included in subject or CA certificates and may be critical or non-critical.

```
id-pkix-authorityInfoAccess OBJECT-IDENTIFIER ::= { pkix 2}

-- authorityInfoAccess ::=  { AuthorityInfoAccessSyntax  }

AuthorityInfoAccessSyntax  ::=  SEQUENCE  {
     certStatus        [0] SEQUENCE OF AccessDescription OPTIONAL,
     certRetrieval     [1] SEQUENCE OF AccessDescription OPTIONAL,
     caPolicy          [2] SEQUENCE OF AccessDescription OPTIONAL,
     caCerts           [3] SEQUENCE OF AccessDescription OPTIONAL  }
```

If certStatus is present, each entry in that sequence describes a mechanism and location for

on-line verification of the status of this certificate, or

the CRL on which this certificate would appear if revoked.

If certRetrieval is present, each entry in the sequence describes how to retrieve all current certificates whose subject is the issuer of the certificate in which this extension appears.

If caPolicy is present, each entry in the sequence describes how to retrieve the policy that was in effect when this certificate was issued.

If caCerts is present, each entry in the sequence describes a mechanism and location for retrieval of certificates the issuer has issued to other CAs.

If the certStatus, certRetrieval, caPolicy, or caCerts sequence has more than one value, conforming applications are not required to process all the values. Successful processing of any one

AccessDescritpion shall be sufficient. It is the responsibility of the certificate issuer to ensure all mechanisms provide the same information.

The expected values for AccessDescription are the values defined in 4.2.2.1. Processing rules for other values for accessMethod are not defined.

If this extension is critical, applications are required to use the information in the certStatus field (if present) to check the revocation status of this certificate, the certRetrieval field (if present) to obtain the issuer's current certificates, and the caCerts field (if present) to obtain certificates issued by the subject to other CAs.

There are no additional processing requirements for cAPolicy if the extension is marked as critical.

4.2.2.3 CA Information Access

Where the subject of a certificate is a CA, the subjectInfoAccess extension may be insufficient. The CA information access extension indicates how to access CA information and services for the subject of the certificate in which the extension appears. Information and services include certificate status or on-line validation services, certificate retrieval, CA policy data, and CA certificates (certificates certifying the target CA to aid in cert path navigation). This extension is syntactically identical to authorityInfoAccess, but is identified by a different OID. This extension may be included only in CA certificates and may be critical or non-critical. CA certificates may include both an authority and a caInfoAccess extension to describe access methods for both the CA and its issuer.

```
id-pkix-caInfoAccess OBJECT-IDENTIFIER ::= { pkix 3 }

-- caInfoAccess ::=  { AuthorityInfoAccessSyntax  }
```

If certStatus is present, each entry in that sequence describes a mechanism and location for

- on-line verification of the status of this certificate, or
- CRL issued by the subject of this certificate.

If certRetrieval is present, each entry in the sequence describes how to retrieve all current certificates whose subject is the subject of the certificate in which this extension appears.

If caPolicy is present, each entry in the sequence describes how to retrieve the current policy tssociated with the subject of this certificate.

If caCerts is present, each entry in the sequence describes a mechanism and location for retrieval of certificates the subject has issued to other CAs.

If the certStatus, certRetrieval, caPolicy, or caCerts sequence has more than one value, conforming applications are not required to process all the values. Successful processing of any one AccessDescritpion shall be sufficient. It is the responsibility of the certificate issuer to ensure all mechanisms provide the same information.

The legal values for AccessDescription shall be as defined in 4.2.2.1.

If this extension is critical, applications are required to use the information in the certStatus field (if present) to check the revocation status of this certificate, the certRetrieval field (if present) to obtain the subject's other current certificates, and the caCerts field (if present) to obtain certificates issued by the subject to other CAs.

There are no additional processing requirements for cAPolicy if the extension is marked as critical.

4.3 Examples

<< Certificate samples including descriptive text and ASN.1 encoded blobs will be inserted. >>

4.3.1 Simple certificate, no extensions

<<TBD>>

4.3.2 Certificate with Private extensions

<<TBD>>

4.3.3 certificate with no subject DN

<<TBD>>

5 CRL and CRL Extensions Profile

As described 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. Emphasis is placed on support for X.509 v2 CRLs. The profile defines a baseline set of information that can be expected in every CRL. Also, the profile defines common locations within the CRL for frequently used attributes, and common representations for these attributes.

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 status mechanisms are provided. Conforming CAs that issue CRLs are required to issue version 2 CRLs. Conforming applications are required to process version 1 and 2 certificates.

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,
     signature            BIT STRING  }

TBSCertList  ::=  SEQUENCE  {
     version                 Version OPTIONAL,
                                  -- if present, must be v2
     signature               AlgorithmIdentifier,
     issuer                  Name,
     thisUpdate              ChoiceOfTime,
     nextUpdate              ChoiceOfTime,
     revokedCertificates     SEQUENCE OF SEQUENCE  {
```

```
            userCertificate         CertificateSerialNumber,
            revocationDate          ChoiceOfTime,
            crlEntryExtensions      Extensions OPTIONAL
                                            -- if present, must be v2
                                  }  OPTIONAL,
      crlExtensions           [0]   Extensions OPTIONAL
                                            -- if present, must be v2
                                  }

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

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

   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

   CertificateSerialNumber  ::=  INTEGER

   Extensions  ::=  SEQUENCE OF Extension

   Extension  ::=  SEQUENCE  {
        extnId                  OBJECT IDENTIFIER,
        critical                BOOLEAN DEFAULT FALSE,
        extnValue               OCTET STRING  }
                                     -- contains a DER encoding of a value
                                     -- of the type registered for use with
                                     -- the extnId object identifier value
```

The following items describe the proposed 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 are described in detail in the following subsections

5.1.1.1 tbsCertList

The first field in the sequence is the tbsCertList. This is a itself a sequence, and is generally thought of as the X.509 CRL. It contains the names of the subject and issuer, a public key associated with the subject an expiration date, and other associated information. The

fields of the basic tbsCertificate are described in detail in section 4.1.2; the tbscertificate may also include extensions which are described in section 4.2.

5.1.1.2 signatureAlgorithm

The signatureAlgorithm field contains the algorithm identifier for the algorithm used by the CA to sign the CertificateList. Section 7.2 lists the supported signature algorithms.

5.1.1.3 signature

The signature field contains a digital signature computed upon the ASN.1 DER encoded TBSCertList. The ASN.1 DER encoded TBSCertificate is used as the input to a one-way hash function. The one-way hash function output value is ASN.1 encoded as an OCTET STRING and the result is encrypted (e.g., using RSA Encryption) to form the signed quantity. This signature value is then ASN.1 encoded as a BIT STRING and included in the Certificate's signature field.

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 CA will issue the next CRL.

Optional fields include lists of revoked certificates and CRL extensions. The revoked certificate list is optional to support the special case where a CA has not revoked any unexpired certificates it has issued. It is expected that nearly all CRLs issued in the Internet PKI will contain one or more lists of revoked certificates. Similarly, the profile requires conforming CAs to use of one CRL extension (CRL number) in all CRLs issued.

5.1.2.1 Version

This field describes the version of the encoded CRL. When extensions are used, as expected in this profile, use version 2 (the integer value is 1). If neither CRL extensions nor CRL entry extensions are present, version 1 CRLs are recommended (e.g., the integer value should be omitted).

5.1.2.2 Signature

This field contains the algorithm identifier for the algorithm used to sign the CRL. Section 7.2 lists the signature algorithms used in

the Internet PKI.

5.1.2.3 Issuer Name

The issuer name identifies the entity who has signed (and issued the CRL). The issuer identity may be carried in the issuer name field and/or the issuerAltName extension. If identity information is present only in the issuerAltName extension, then the issuer name may be an empty sequence and the issuerAltName extension must be critical.

5.1.2.4 This Update

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

CAs conforming to this profile shall not issue CRLs where thisUpdate is encoded as GeneralizedTime before the year 2005. CAs conforming to this profile shall not issue CRLs where thisUpdate is encoded as UTCTime after the year 2015.

Where encoded as UTCTime, thisUpdate shall be specified and interpreted as defined in Section 4.1.2.5.1. Where encoded as GeneralizedTime, thisUpdate shall 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. nextUpdate may be encoded as UTCTime or GeneralizedTime.

CAs conforming to this profile shall not issue CRLs where nextUpdate is encoded as GeneralizedTime before the year 2005. CAs conforming to this profile shall not issue CRLs where nextUpdate is encoded as UTCTime after the year 2015.

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

5.1.2.6 Revoked Certificates

Revoked certificates are listed. The revoked certificates are named by their serial numbers. Certificates are uniquely identified by the combination of the issuer name or issuer alternative name along with

the user certificate serial number. The date on which the revocation occured is specified. The time for revocationDate shall 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.

5.2 CRL Extensions

The extensions defined by ANSI X9 and ISO for X.509 v2 CRLs [X.509-AM] [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 an critical extension which it does not know how to process. However, an unrecognized non-critical extension may be ignored. The following presents 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 CAs that issue CRLs are required to support the CRL number extension (5.2.3), and include it in all CRLs issued. Conforming applications are required to support the critical and optionally critical CRL extensions issuer alternative name (5.2.2), issuing distribution point (5.2.4) and delta CRL indicator (5.2.5).

5.2.1 Authority Key Identifier

The authority key identifier extension provides a means of identifying the particular public 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. The key identifier method is recommended in this profile. This extension would be used where an issuer has multiple signing keys, either due to multiple concurrent key pairs or due to changeover. In general, this non-critical extension should be included in certificates.

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 shall be used.

Further, if the only issuer identity included in the CRL is an alternative name form (e.g., an electronic mail address), then the issuer distinguished name should be empty (an empty sequence), the issuerAltName extension should be used, and the issuerAltName extension must be marked critical. If more than one issuerAltName extension appears in the CRL and the issuer distinguished name is empty, exactly one issuerAltName extension must be marked critical.

The object identifier 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 increacing sequence number for each CRL issued by a given CA through a specific CA X.500 Directory entry or CRL distribution point. This extension allows users to easily determine when a particular CRL supercedes another CRL. CAs conforming to this profile shall include this extension in all CRLs.

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

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

5.2.4 Issuing Distribution Point

The issuing distribution point is a critical CRL extension that identifies the CRL distribution point for a particular CRL, and it indicates whether the CRL covers revocation for end entity certificates only, CA certificates only, or a limitied set of reason codes. Since this extension is critical, all certificate users must be prepared to receive CRLs with this extension.

The CRL is signed using the CA'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 that the Directory entry of the CA.

CRL distribution points, if used by a CA, should be partition the CRL on the basis of compromise and routine revocation. That is, the revocations with reason code keyCompromise (1) shall appear in one distribution point, and the revocations with other reason codes shall appear in another distribution point.

```
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 }
```

5.2.5 Delta CRL Indicator

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

When a delta-CRL is issued, the CAs shall also issue a complete CRL.

The value of BaseCRLNumber identifies the CRL number of the base CRL that was used as the starting point in the generation of this delta-CRL. The delta-CRL contains the changes between the base CRL and the current CRL issued along with the delta-CRL. It is the decision of a CA as to whether to provide delta-CRLs. Again, a delta-CRL shall not be issued without a corresponding CRL. The value of CRLNumber for both the delta-CRL and the corresponding CRL shall be identical.

A CRL user constructing a locally held CRL from delta-CRLs shall consider the constructed CRL incomplete and unusable if the CRLNumber of the received delta-CRL is more that one greater that the CRLnumber of the delta-CRL last processed.

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

deltaCRLIndicator ::= BaseCRLNumber

BaseCRLNumber ::= CRLNumber
```

5.3 CRL Entry Extensions

The CRL entry extensions already defined by ANSI X9 and ISO for X.509 v2 CRLs [X.509-AM] [X9.55] provide methods for associating additional attributes with CRL entries. 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

presents 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 are non-critical; support for these extensions is optional for conforming CAs and applications. However, CAs that issue CRLs are strongly encouraged to include reason codes (5.3.1) 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. CAs are strongly encouraged to include 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) }
```

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 shall 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 a id-holdinstruction-callissuer must call the certificate issuer or reject the certificate. Conforming applications which encounter a id-holdinstruction-reject ID shall reject the transaction. id-holdinstruction-none is semantically equivalent to the absence of a holdInstructionCode. Its use is strongly deprecated for the Internet PKI.

<<Note: I didn't think id-holdinstruction-pickupToken was appropriate 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, but it must be later than the issue date of the previously issued CRL. Remember that the revocation date in the CRL entry specifies the date that the CA revoked the certificate. Whenever this information is available, CAs are strongly encouraged to share it with CRL users.

The GeneralizedTime values included in this field shall be expressed in Greenwich Mean Time (Zulu) and omit trailing zeros in fractional seconds. GeneralizedTime shall be expressed as YYYYMMDDHHMMSSZ.

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

invalidityDate ::=  GeneralizedTime
```

5.4 Examples

5.4.1 Empty CRL

<<TBD>>

5.4.2 CRL with entries, no extensions

<<TBD>>

5.4.3 CRL with extensions

<<TBD>>

6 Certificate Path Validation

Certification path validation procedures for the Internet PKI are based on Section 12.4.3 of [X.509-AM].

Certification path processing verifies the binding between the subject distinguished name and subject public key. The basic constraints and policy constraints extensions facilitate automated, self-contained implementation of certification path processing logic.

The following is an outline of a procedure for validating certification paths. An implementation shall be functionally equivalent to the external behaviour resulting from this procedure. Any algorithm may be used by a particular implementation so long as it derives the correct result.

The inputs to the certification path processing procedure are:

(a) a set of certificates comprising a certification path;

(b) a CA name and trusted public key value (or an identifier of such a key if the key is stored internally to the certification path processing module) for use in verifying the first certificate in the certification path;

(c) a set of initial-policy identifiers (each comprising a sequence of policy element identifiers), which identifies one or more certificate policies, any one of which would be acceptable for the purposes of certification path processing; and

(d) the current date/time (if not available internally to the certification path processing module).

The outputs of the procedure are:

(a) an indication of success or failure of certification path validation;

(b) if validation failed, a reason for failure; and

(c) if validation was successful, a (possibly empty) set of policy qualifiers obtained from CAs on the path.

The procedure makes use of the following set of state variables:

(a) acceptable policy set: A set of certificate policy identifiers comprising the policy or policies recognized by the public key user together with policies deemed equivalent through policy mapping;

(b) constrained subtrees: A set of root names defining a set of subtrees within which all subject names in subsequent certificates in the certification path shall fall; if no restriction is in force this state variable takes the special value unbounded; and

(c) excluded subtrees: A set of root names defining a set of subtrees within which no subject name in subsequent certificates in the certification path may fall; if no restriction is in force this state variable takes the special value empty.

The procedure involves an initialization step, followed by a series of certificate-processing steps. The initialization step comprises:

(a) Initialize the constrained subtress to unbounded;

(b) Initialize the excluded subtrees indicator to empty; and

(c) Initialize the acceptable policy set to the set of initial-policy identifiers.

Each certificate is then processed in turn, starting with the certificate signed using the trusted CA public key which was input to this procedure. The last certificate is processed as an end-entity certificate; all other certificates (if any) are processed as CA-certificates.

The following checks are applied to all certificates:

(a) Check that the signature verifies, that dates are valid, that the subject and issuer names chain correctly, and that the certificate has not been revoked;

If the certificate has an empty sequence in the name field, name chaining will use the critical altSubjectNames and altIssuerNames fields. If the certificate has a critical authorityInfoAccess or caInfoAccess extension, the information in that extension must be used to determine the status of the certificates.

(b) If a key usage restriction extension is present in the certificate and contains a certPolicySet component, check that at least one member of the acceptable policy set appears in the field;

(c) Check that the subject name or critical AltSubjectName extension is consistent with the constrained subtrees state variables; and

(d) Check that the subject name or critical AltSubjectName extension is consistent with the excluded subtrees state variables.

If any one of the above checks fails, the procedure terminates, returning a failure indication and an appropriate reason. If none of the above checks fail on the end-entity certificate, the procedure terminates, returning a success indication together with the set of all policy qualifier values encountered in the set of certificates.

For a CA-certificate, the following constraint recording actions are then performed, in order to correctly set up the state variables for the processing of the next certificate:

(a) If permittedSubtrees is present in the certificate, set the constrained subtrees state variable to the intersection of its previous value and the value indicated in the extension field.

(b) 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.

Note: It is possible to specify an extended version of the above certification path processing procedure which results in default behaviour identical to the rules of Privacy Enhanced Mail [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.

7 Algorithm Support

This section describes cryptographic alogrithms which may be used with this standard. The section describes one-way hash functions and digital signature algorithms which may be used to sign certificates and CRLs, and identifies object identifiers for public keys contained in a certificate.

Conforming CAs and applications are not required to support the algorithms or algorithm identifiers described in this section. However, this profile requires conforming CAs and applications to conform when they use the algorithms identified here.

7.1 One-way Hash Functions

This section identifies one-way hash functions for use in the Internet PKI. One-way hash functions are also called message digest algorithms. SHA-1 is the preferred one-way hash function for the Internet PKI. However, PEM uses MD2 for certificates [RFC 1422] [RFC 1423]. For this reason, MD2 is included in this profile.

7.1.1 MD2 One-way Hash Function

MD2 was developed by Ron Rivest, but RSA Data Security has not placed the MD2 algorithm in the public domain. Rather, RSA Data Security has granted license to use MD2 for non-commerical Internet Privacy-Enhanced Mail. For this reason, MD2 may continue to be used with PEM certificates, but SHA-1 is preferred. MD2 is fully described in RFC 1319 [RFC 1319].

At the Selected Areas in Cryptography '95 conference in May 1995, Rogier and Chauvaud presented an attack on MD2 that can nearly find collisions [RC95]. Collisions occur when two different messages generate the same message digest. A checksum operation in MD2 is the only remaining obstacle to the success of the attack. For this reason, the use of MD2 for new applications is discouraged. It is still reasonable to use MD2 to verify existing signatures, as the ability to find collisions in MD2 does not enable an attacker to find new messages having a previously computed hash value.

<< More information on the attack and its implications can be obtained from a RSA Laboratories security bulletin. These bulletins are available from <http://www.rsa.com/>. >>

7.1.2 SHA-1 One-way Hash Function

SHA-1 was developed by the U.S. Government. SHA-1 is fully described in FIPS 180-1 [FIPS 180-1].

SHA-1 is the one-way hash function of choice for use with both the RSA and DSA signature algorithms.

7.2 Signature Algorithms

Certificates and CRLs described by this standard may be signed with any public key signature algorithm. The certificate or CRL indicates

the algorithm through an algorithmidentifier which appears in the signatureAlgorithm field in a Certificate or CertificateList. This algorithmidentfier is an OID and has optionally associated parameters. This section identifies algorithm identifiers and parameters that shall be used in the signatureAlgorithm field in a Certificate or CertificateList.

RSA and DSA are the most popular signature algorithms used in the Internet. Signature algorithms are always used in conjunction with a one-way hash function identified in Section 7.1.

The signature algorithm (and one-way hash function) used to sign a certificate or CRL is indicated by use of an algorithm identifier. An algorithm identifier is an object identifier, and may include associated parameters. This section identifies OIDS for RSA and DSA and the corresponding parameters.

The data to be signed (e.g., the one-way hash function output value) is first ASN.1 encoded as an OCTET STRING and the result is encrypted (e.g., using RSA Encryption) to form the signed quantity. This signature value is then ASN.1 encoded as a BIT STRING and included in the Certificate or CertificateList (in the signature field).

7.2.1 RSA Signature Algorithm

A patent statement regarding the RSA algorithm can be found at the end of this profile.

The RSA algorithm is named for it's inventors: Rivest, Shamir, and Adleman. The RSA signature algorithm combines either the MD2 or the SHA-1 one-way hash function with the RSA asymmetric encryption algorithm. The RSA signature algorithm with MD2 and the RSA encryption algorithm is defined in PKCS #1 [PKCS#1]. As defined in PKCS #1, the ASN.1 object identifier used to identify this signature algorithm is:

```
md2WithRSAEncryption OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1)
    pkcs-1(1) 2  }
```

The RSA signature algorithm with SHA-1 and the RSA encryption algorithm is defined in by the OSI Ineroperability Workshop in []. As defined in [OIW], the ASN.1 object identifier used to identify this signature algorithm is:

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

When either of these object identifiers is used within the ASN.1 type AlgorithmIdentifier, the parameters component of that type shall be the ASN.1 type NULL.

When signing, the RSA algorithm generates an integer y. This value is converted to a bit string such that the most significant bit in y is the first bit in the bit string and the least significant bit in y is the last bit in the bit string.

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

7.2.2 DSA Signature Algorithm

A patent statement regarding the DSA can be found at the end of this profile.

The Digital Signature Algorithm (DSA) is also called the Digital Signature Standard (DSS). DSA was developed by the U.S. Government, and DSA is used in conjunction with the the SHA-1 one-way hash function. DSA is fully described in FIPS 186 [FIPS 186]. The ASN.1 object identifiers used to identify this signature algorithm are:

```
id-dsa-with-sha1 ID  ::=  {
        iso(1) member-body(2) us(840) x9-57 (10040) secsig(2)
        x9algorithm(4) 3 }
```

The id-dsa-with-sha1 algorithm syntax has NULL parameters. The DSA parameters in the subjectPublicKeyInfo field of the certificate of the issuer shall apply to the verification of the signature.

If the subjectPublicKeyInfo AlgorithmIdentifier field has NULL parameters and the CA signed the subject certificate using DSA, then the certificate issuer's parameters apply to the subject's DSA key. If the subjectPublicKeyInfo AlgorithmIdentifier field has NULL parameters and the CA signed the subject with a signature algorithm other than DSA, then clients shall not validate the certificate.

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

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

7.3 Subject Public Key Algorithms

Certificates described by this standard may convey a public key for any public key algorithm. The certificate indicates the algorithm through an algorithmidentifier. This algorithmidentfieier is an OID and optionally associated parameters.

This section identifies preferred OIDs and parameters for the RSA, DSA, KEA, and Diffie-Hellman algorithms. Conforming CAs shall use the identified OIDs when issuing certificates containing public keys for these algorithms. Conforming applications supporting any of these algorithms shall, at a minimum, recognize the OID identified in this section.

7.3.1 RSA Keys

The object identifier rsaEncryption identifies RSA public keys.

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

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

The rsaEncryption object identifier is intended to be used in the algorithm field of a value of type AlgorithmIdentifier. The parameters field shall have ASN.1 type NULL for this algorithm identifier.

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

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

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

This object identifier is used in public key certificates for both RSA signature keys and RSA encryption keys. The intended application for the key may be indicated in the key usage field (see Section 4.2.1.3). The use of a single key for both signature and encryption

purposes is not recommended, but is not forbidden.

7.3.2 Diffie-Hellman Key Exchange Key

This diffie-hellman object identifier supported by this standard is defined by ANSI X9.42.

```
dhpublicnumber OBJECT IDENTIFIER ::= { iso(1) member-body(2)
          US(840) ansi-x942(10046) number-type(2) 1 }

DHParameter ::= SEQUENCE {
     prime INTEGER, -- p
     base INTEGER, -- g }
```

The dhpublicnumber object identifier is intended to be used in the algorithm field of a value of type AlgorithmIdentifier. The parameters field of that type, which has the algorithm-specific syntax ANY DEFINED BY algorithm, would have ASN.1 type DHParameter for this algorithm.

```
DHParameter ::= SEQUENCE {
  prime INTEGER, -- p
  base INTEGER, -- g }
```

The fields of type DHParameter have the following meanings:

prime is the prime p.

base is the base g.

The Diffie-Hellman public key (an INTEGER) is mapped to a subjectPublicKey (a BIT STRING) as follows: the most significant bit (MSB) of the INTEGER becomes the MSB of the BIT STRING; the least significant bit (LSB) of the INTEGER becomes the LSB of the BIT STRING.

7.3.3 DSA Signature Keys

The object identifier supported by this standard is

```
id-dsa ID ::= { iso(1) member-body(2) us(840) x9-57(10040)
          secsig(2) x9algorithm(4) 1 }
```

The id-dsa algorithm syntax includes optional parameters. These parameters are commonly referred to as p, q, and g. If the DSA algorithm parameters are absent from the subjectPublicKeyInfo AlgorithmIdentifier and the CA signed the subject certificate using DSA, then the certificate issuer's DSA parameters apply to the

subject's DSA key. If the DSA algorithm parameters are absent from the subjectPublicKeyInfo AlgorithmIdentifier and the CA signed the subject certificate using a signature algorithm other than DSA, then the subject's DSA parameters are distributed by other means. The parameters are included using the following ASN.1 structure:

```
Dss-Parms  ::=  SEQUENCE  {
    p             INTEGER,
    q             INTEGER,
    g             INTEGER  }
```

If the subjectPublicKeyInfo AlgorithmIdentifier field has NULL parameters and the CA signed the subject certificate using DSA, then the certificate issuer's parameters apply to the subject's DSA key. If the subjectPublicKeyInfo AlgorithmIdentifier field has NULL parameters and the CA signed the subject with a signature algorithm other than DSA, then clients shall not validate the certificate.

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

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

The encoded signature is conveyed as the value of the BIT STRING signature in a Certificate or CertificateList.

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

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

7.3.4 Key Exchange Algorithm (KEA)

The Key Exchange Algorithm (KEA) is a classified algorithm for exchanging keys. A KEA "pairwise key" may be generated between two users if their KEA public keys were generated with the same KEA parameters. The KEA parameters are not included in a certificate; instead a "domain identifier" is supplied in the parameters field.

When the subjectPublicKeyInfo field contains a KEA key, the algorithm identifier and parameters shall be as defined in [sdn.701r]:

```
   id-keyEncryptionAlgorithm  OBJECT IDENTIFIER   ::=
          { 2 16 840 1 101 2 1 1 22 }

   KEA-Parms-Id     ::= OCTET STRING
```

The Kea-Parms-Id shall always appear when the subjectPublicKeyInfo field algorithm identifier is id-keyEncryptionAlgorithm. Kea-Parms-Id is the "domain identifier" and is ten octets in length. If the Kea-Parms-Id of two KEA keys are equivalent, the subjects possess the same KEA parameter values and may exchange keys.

<<Need encoding of KEA key>>

8. ASN.1 Structures and OIDs

```
PKIX1 DEFINITIONS ::=

BEGIN


-- need ASN.1 for:
-- AlgorithmIdentifier
-- ORAddress


-- attribute data types --

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

AttributeType           ::=     OBJECT IDENTIFIER

AttributeValue          ::=     ANY

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

AttributeValueAssertion ::=     SEQUENCE {AttributeType, AttributeValue}

-- 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..maxSize)),
           printableString         PrintableString (SIZE (1..maxSize)),
           universalString         ANY -- the '93 ASN.1 type UniversalString
                                       }

   -- basic stuff starts 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      CertificateValidityDate,
        notAfter       CertificateValidityDate }

   CertificateValidityDate ::= CHOICE {
```

```
        utcTime        UTCTime,
        generalTime    GeneralizedTime }

   UniqueIdentifier  ::=  BIT STRING

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

   Extensions  ::=  SEQUENCE OF Extension

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

   -- Extension ::= { {id-ce 15}, ... , keyUsage }

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

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

   -- authorityKeyIdentifier ::= AuthorityKeyIdentifier

   KeyIdentifier ::= OCTET STRING

   -- subjectKeyIdentifier ::= KeyIdentifier

   KeyUsage ::= BIT STRING {
        digitalSignature        (0),
        nonRepudiation          (1),
        keyEncipherment         (2),
        dataEncipherment        (3),
```

```
        keyAgreement            (4),
        keyCertSign             (5),
```

```
        cRLSign                (6) }

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

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

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

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

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

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

   PolicyQualifierId ::= ENUMERATED {
           qualId1 (1), qualId2 (2), qualId3 (3), qualId4 (4), qualId5 ( 5 ) }

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

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

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

   SubjectAltName ::= GeneralNames
```

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

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

   -- OTHER-NAME ::= TYPE-IDENTIFIER  note: not supported in '88 ASN.1
   --              substituted ANY where used [GeneralName otherName]

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

   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

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

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

   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)
```

```
   id-ce-policyConstraints OBJECT IDENTIFIER ::=  { id-ce 34 }

   PolicyConstraints ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        policySet                       [0] CertPolicySet OPTIONAL,
        requireExplicitPolicy           [1] SkipCerts OPTIONAL,
        inhibitPolicyMapping            [2] SkipCerts OPTIONAL }

   SkipCerts ::= INTEGER (0..MAX)

   CertPolicySet ::= SEQUENCE SIZE (1..MAX) OF CertPolicyId

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

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

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

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

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

   pkix  OBJECT IDENTIFIER ::= { 1 3 6 1 5 3 }

   -- Object identifiers for ftp, http, smtp and ldap protocols

   applTCPProto OBJECT IDENTIFIER ::= { 1 3 6 1 2 1 27 4 }

   ftpID    OBJECT-IDENTIFIER ::= {applTCPProtoID 21}
   httpID   OBJECT-IDENTIFIER ::= {applTCPProtoID 80}
   smtpID   OBJECT-IDENTIFIER ::= {applTCPProtoID 25}
   ldapID   OBJECT-IDENTIFIER ::= {applTCPProtoID 389}

   -- Object identifier for the X.500 directory access protocol

   dap OBJECT-IDENTIFIER ::= { 2 5 3 1 }
```

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

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

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

   id-pkix-authorityInfoAccess OBJECT IDENTIFIER ::= { pkix 2 }

   AuthorityInfoAccessSyntax  ::=  SEQUENCE  {
        certStatus        [0] SEQUENCE OF AccessDescription OPTIONAL,
        certRetrieval     [1] SEQUENCE OF AccessDescription OPTIONAL,
        caPolicy          [2] SEQUENCE OF AccessDescription OPTIONAL,
        caCerts           [3] SEQUENCE OF AccessDescription OPTIONAL  }

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

   -- caInfoAccess ::=  {
   --     AuthorityInfoAccessSyntax  }

   -- 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              ChoiceOfTime,
        nextUpdate              ChoiceOfTime,
        revokedCertificates     SEQUENCE OF SEQUENCE  {
             userCertificate         CertificateSerialNumber,
             revocationDate          ChoiceOfTime,
             crlEntryExtensions      Extensions OPTIONAL
                                           -- if present, must be v2
                                   }  OPTIONAL,
        crlExtensions           [0]  Extensions OPTIONAL
                                           -- if present, must be v2
                                   }
```

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

   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

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

   CertificateSerialNumber  ::=  INTEGER

   Extensions  ::=  SEQUENCE OF Extension

   Extension  ::=  SEQUENCE  {
        extnId                  OBJECT IDENTIFIER,
        critical                BOOLEAN DEFAULT FALSE,
        extnValue               OCTET STRING  }
                                   -- contains a DER encoding of a value
                                   -- of the type registered for use with
                                   -- the extnId object identifier value

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

   CRLNumber ::= INTEGER (0..MAX)

   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 }


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

   -- deltaCRLIndicator ::= BaseCRLNumber

   BaseCRLNumber ::= CRLNumber

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

   -- reasonCode EXTENSION ::= {
```

```
   --      SYNTAX  CRLReason
   --      IDENTIFIED BY { id-ce 21 } }

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

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

   HoldInstructionCode ::= OBJECT IDENTIFIER

   member-body ID ::= { iso 2 }
   us ID ::= { member-body 840 }
   x9cm ID ::= { us 10040 }
   holdInstruction ID ::= {x9cm 2}

   id-holdinstruction-none ID ::= {holdInstruction 1}
   id-holdinstruction-callissuer ID ::= {holdInstruction 2}
   id-holdinstruction-reject ID ::= {holdInstruction 3}

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

   InvalidityDate ::=  GeneralizedTime

   -- Algorithm strustures

        md2WithRSAEncryption OBJECT IDENTIFIER  ::=  {
            iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1)
            pkcs-1(1) 2  }


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

           id-dsa-with-sha1 ID  ::=  {
                   iso(1) member-body(2) us(840) x9-57 (10040) secsig(2)
                   x9algorithm(4) 3 }

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

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

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

      dhpublicnumber OBJECT IDENTIFIER ::= { iso(1) member-body(2)
                US(840) ansi-x942(10046) 1 }

      DHParameter ::= SEQUENCE {
           prime INTEGER, -- p
           base INTEGER -- g
                }

      id-dsa ID ::= { iso(1) member-body(2) us(840) x9-57(10040)
                secsig(2) x9algorithm(4) 1 }

      Dss-Parms  ::=  SEQUENCE  {
          p             INTEGER,
          q             INTEGER,
          g             INTEGER  }

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

   id-keyEncryptionAlgorithm  OBJECT IDENTIFIER   ::=
          { 2 16 840 1 101 2 1 1 22 }

   KEA-Parms-Id     ::= OCTET STRING

  id-ce-authorityKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 35 }
  id-ce-subjectKeyIdentifier OBJECT IDENTIFIER ::=  { id-ce 14 }
  id-ce-keyUsage OBJECT IDENTIFIER ::=  { id-ce 15 }
  id-pkix-policy-CPS OBJECT IDENTIFIER ::= { pkix 4 }

  CPSuri ::= IA5String

  id-pkix-policy-userNotice OBJECT IDENTIFIER ::= { pkix 5 }

  UserNotice ::= CHOICE {
    ia5String     IA5String,
    bnpString     ANY -- defined as BMPString in '93 ASN.1
                        }

  END
```

References

[X9.57] ANSI X9.57

[FIPS 180-1] Federal Information Processing Standards Publication (FIPS PUB) 180-1, Secure Hash Standard, 17 April 1995. [Supersedes FIPS PUB 180 dated 11 May 1993.]

[FIPS 186] Federal Information Processing Standards Publication (FIPS PUB) 186, Digital Signature Standard, 18 May 1994.

[PKCS#1] PKCS #1: RSA Encryption Standard, Version 1.4, RSA Data Security, Inc., 3 June 1991.

[RC95] Rogier, N. and Chauvaud, P., "The compression function of MD2 is not collision free," Presented at Selected Areas in Cryptography '95, Carleton University, Ottawa, Canada, 18-19 May 1995.

[RFC 1319] Kaliski, B., "The MD2 Message-Digest Algorithm," RFC 1319, RSA Laboratories, April 1992.

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

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

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

[SDN.701R] SDN.701, "Message Security Protocol", Revision 4.0 1996-06-07 with "Corrections to Message Security Protocol, SDN.701, Rev 4.0, 96-06-07." August 30, 1996.

[X.208] << Do we want to reference the 1988 or 1993 version? >>

[X.509-AM] << Need final reference >>

[X9.55] << Need final reference >>

Patent Statements

The Internet PKI relies on the use of patented public key technology. The Internet Standards Process as defined in RFC 1310 requires a written statement from the Patent holder that a license will be made available to applicants under reasonable terms and conditions prior to approving a specification as a Proposed, Draft or Internet

Standard.

Patent statements for DSA, RSA, and Diffie-Hellman follow. These statements have been supplied by the patent holders, not the authors of this profile.

Digital Signature Algorithm (DSA)

The U.S. Government holds patent 5,231,668 on the Digital Signature Algorithm (DSA), which has been incorporated into Federal Information Processing Standard (FIPS) 186. The patent was issued on July 27, 1993.

The National Institute of Standards and Technology (NIST) has a long tradition of supplying U.S. Government-developed techniques to committees and working groups for inclusion into standards on a royalty-free basis. NIST has made the DSA patent available royalty-free to users worldwide.

Regarding patent infringement, FIPS 186 summarizes our position; the Department of Commerce is not aware of any patents that would be infringed by the DSA. Questions regarding this matter may be directed to the Deputy Chief Counsel for NIST.

RSA Signature and Encryption

<< Now that PKP has dissolved, a revised patent statement for RSA from RSADSI is needed. >>

Diffie-Hellman Key Agreement

<< Now that PKP has dissolved, a revised patent statement for Diffie-Hellman from Cylink is needed. >>

Obsolete PKP Patent Statement

<< This statement is included here until a replacement from RSADSI and Cylink can be obtained. >>

The Massachusetts Institute of Technology and the Board of Trustees of the Leland Stanford Junior University have granted Public Key Partners (PKP) exclusive sub-licensing rights to the following patents issued in the United States, and all of their corresponding foreign patents:

Cryptographic Apparatus and Method
("Diffie-Hellman")..... No. 4,200,770

Public Key Cryptographic Apparatus
and Method ("Hellman-Merkle")..... No. 4,218,582

Cryptographic Communications System and
Method ("RSA")..... No. 4,405,829

Exponential Cryptographic Apparatus
and Method ("Hellman-Pohlig")..... No. 4,424,414

These patents are stated by PKP to cover all known methods of practicing the art of Public Key encryption, including the variations collectively known as El Gamal.

Public Key Partners has provided written assurance to the Internet Society that parties will be able to obtain, under reasonable, nondiscriminatory terms, the right to use the technology covered by these patents. This assurance is documented in RFC 1170 titled "Public Key Standards and Licenses". A copy of the written assurance dated April 20, 1990, may be obtained from the Internet Assigned Number Authority (IANA).

The Internet Society, Internet Architecture Board, Internet Engineering Steering Group and the Corporation for National Research Initiatives take no position on the validity or scope of the patents and patent applications, nor on the appropriateness of the terms of the assurance. The Internet Society and other groups mentioned above have not made any determination as to any other intellectual property rights which may apply to the practice of this standard. Any further consideration of these matters is the user's own responsibility.

Security Considerations

This entire memo is about security mechanisms.

Author Addresses:

Russell Housley
SPYRUS
PO Box 1198
Herndon, VA 20172
USA
housley@spyrus.com

Warwick Ford
VeriSign, Inc.
One Alewife Center
Cambridge, MA 02140
wford@verisign.com

Tim Polk
NIST
Building 820, Room 426
Gaithersburg, MD 20899
wpolk@nist.gov

David Solo
BBN
150 CambridgePark Drive
Cambridge, MA 02140
USA
solo@bbn.com