PKIX Working Group R. Housley (SPYRUS)
Internet Draft W. Ford (NorTel)
S. Farrell (SSE)
D. Solo (BBN)
expires in six months November 1995

Internet Public Key Infrastructure

draft-ietf-pkix-ipki-00.txt

Status of this Memo

Abstract

This is the first draft of the Internet Public Key Infrastructure. It is provided as a strawman for the first meeting of the PKIX Working Group. The intent of this strawman is to generate productive discussions at the first meeting.

1 Executive Summary

<< Write this last. >>

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 HTTP, electronic mail, IPSP, user authentication, electronic payment systems, as well as others. In order to relieve some of the obstacles to using X.509 certificates, this draft will define profiles, rules, and management protocols that should serve to promote the development of reusable certificate management systems; development of reusable application tools; and interoperabilty determined by policy, not syntax.

Many communities will need to supplement, or possibly replace, this profile in order to meet the requirements of specialized domains or environments with additional authorization, assurance, or operational requirements. However, for basic applications, it is essential that a core of features be defined and that common means of representing common information be agreed to so that application developers can obtain necessary information without regard to the issuer of a particular certificate.

As supplemental authorization and attribute management tools emerge, such as attribute certificates, it may be appropriate to limit what the certificate is used for in terms of conveying authenticated attributes as opposed to other means of conveying information.

<< Note, this section needs to be expanded >>

2.1 Communication and Topology

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

2.2 Access Control and Acceptability Decisions

The goal of the Public Key Infrstructure (PKI) is to meet the needs of deterministic, automated access control and authorization functions. This will drive the types of attributes and the nature of the identity 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

In this context, user refers to the users of the client software and the subjects of the certificates. These are the readers and writers of electronic mail, the clients for WWW browsers, etc. A goal of this profile is to recognize the limitations of both the platforms

these users will employ and the sophistication/attentiveness of the users. This manifests itself in requirements to simplify the configuration responsibility of the user (e.g., root keys, rules), make platform usage constraints explicit in the certificate, to construct certification path constraints which shield the user from malicious action, and to construct applications which sensibly automate checking functions.

2.4 Administration Expectations

As with users, the certificate profile should also be structured to be consistent with the types of individuals who must administer the CA space. Providing such an administrator with unbounded choices complicates not only the software that must process these certificates but also increases the chances that a subtle mistake by the CA administrator will result in broader compromise.

3 Overview of Approach

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, and are not applicable to public key infrastructures.

The Internet Privacy Enhanced Mail (PEM) proposals, published in 1993, included specifications for a public key infrastructure based on X.509 version 1 certificates [RFC 1422]. The experience gained in attempts to deploy RFC 1422 made it clear that the v1 and v2

certificate formats were deficient in several respects. Most importantly, more fields were needed to carry information which PEM design and implementation experience had proven necessary. In response to these new requirements, ISO/IEC 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 having a need. In August, 1995, standardization of the basic v3 format was completed [ISO TC].

ISO/IEC and ANSI X9 have also developed a set of standard extensions for use in the v3 extensions field [ISO DAM]. 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 profiles of use of the X.509 v3 extensions tailored for the Internet. It is one goal of this document to specify such profiles.

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 ertification 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 a 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 elimiates 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 document will also consider standard approaches to on-line revocation notification.

3.4 Supporting 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.

Section 9 defines a set of standard protocols supporting the above functions. The protocols for conveying these exchanges in different environments (on-line, E-mail, and WWW) are specified in Section 10.

4 Certificate and Certificate Extensions Profile

As described above, one goal of this draft is to create a profile for X.509 v3 certificates that will foster interoperability and a reusable public key infrastructure. To achieve this goal, some assumptions need to be made about the nature of information to be included along with guidelines for how extensibility will be employed.

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 draft 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. The draft will define a baseline set of information along with common locations within a certificate and common representations for common information. Environments with additional requirements may build on this profile or may replace it.

4.1 Basic Certificate Fields

The X.509 v3 certificate Basic syntax is as follows. For signature calculation, the certificate is ASN.1 DER encoded (reference). ASN.1 DER encoding is a tag, length, value encoding system for each element.

```
Certificate  ::=  SIGNED  {  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            UTCTime,
     notAfter             UTCTime  }

UniqueIdentifier  ::=  BIT STRING

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

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

4.1.1 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 UniqueID 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).

<< All capabilites available in X.509 v2 certificates are available in X.509 v3 certificates. Since there are so few X.509 v2 certificate implementations, should the profile prohibit the use of v2? >>

4.1.2 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).

<< Do we want to define a maximum value for the serial number? >>

4.1.3 Signature

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

4.1.4 Issuer Name

The issuer name provides a globally unique identifier of the authority signing the certificate. The syntax of the issuer name is an X.500 distinguished name. A name in the certificate may provide semantic information, may provide a reference to an external information store or service, provides a unique identifier, may

provide authorization information, or may provide a basis for managing the CA relationships and certificate paths (other purposes are also possible). This strawman suggests that the issuer (and subject) name fields must provide a globally unique identifier. In addition, they should contain semantic information identifying the issuer/subject (e.g. a full name, organization name, etc.). Access information will be provided in a separate extension (when other than via X.500 directory) and internet specific identities (electronic mail address, DNS name, and URLs) will be carried in alternative name extensions.

<< Further discussion of naming guidelines for internet use is needed. >>

4.1.5 Validity

This field indicates the dates on which the certificate becomes valid (notBefore) and on which the certificate ceases to be valid (notAfter).

4.1.6 Subject Name

The purpose of the subject name is to provide a unique identifier of the subject of the certificate. The syntax of the subject name is an X.500 distinguished name. The discussion in section 4.1.4 on issuer names applies to subject names as well.

4.1.7 Subject Public Key Info

This field is used to carry the public key and identify the algorithm with which the key is used.

4.1.8 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. Based on the approach to naming, names will not be reused and internet certificates will not make use of these unique identifiers.

4.2 Certificate Extensions

The extensions already defined by ANSI X9 and ISO for X.509 v3 certificates provide methods for associating additional attributes with users or public keys and for managing the certification hierarchy. The X.509 v3 certificate format also allows communities to define private extensions to carry information unique to those communities. Each extension in a certificate 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.

4.2.1 Subject Alternative Name

The altNames 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, and a URL. Each of these are IA5 strings. Multiple instances may be included. Whenever such identities are to be bound in a certificate, the subject alternative name (or issuer alternative name) field shall be used.

<< This implies that encoding of such identities within the subject or issuer distinguished name is discouraged. >>

<< Note definition is based on a recommended change to the DAM. >>

```
AltNames  ::=  SEQUENCE OF GeneralName

GeneralName  ::=  CHOICE  {
     otherName       [0] INSTANCE OF OTHER-NAME,
     rfc822Name      [1] IA5String,
     dNSName         [2] IA5String,
     x400Address     [3] ORAddress,
     directoryName   [4] Name,
     ediPartyName    [5] IA5String,
     url             [6] IA5String }
```

<< Should we permit an IP address? With the current list of choices, IPSec would use dnsName. This leads to trusted resolution of DNS Names to IP Addresses which is not done today. Maybe IP address is too specific and LAN address should be allowed too. >>

4.2.2 Issuer Alternative Name

As with 4.2.1, this extension is used to bind Internet style identities to the issuer name.

4.2.3 Certificate Policies

The certificatePolicies extension contains an object identifier (OID) which indicates the policy under which the certificate has been issued. Use of policies is discussed elsewhere in this draft.

4.2.4 Key Attributes

The keyAttributes extension contains information about the key itself including a unique key identifier, a key usage period (lifetime of the key as opposed to the lifetime of the certificate), and key usage. The Internet certificate should use the keyAttributes extension and contain a key identifier and private key validity to aid in system management. The key usage field in this extension is intended to be advisory (as contrasted with the key usage restriction extension which imposes mandatory restrictions). The key usage field in this extension should not be used.

```
KeyAttributes  ::=  SEQUENCE  {
     keyIdentifier           KeyIdentifier OPTIONAL,
     intendedKeyUsage        KeyUsage  OPTIONAL,
     privateKeyUsagePeriod   PrivateKeyValidity OPTIONAL  }

KeyIdentifier  ::=  OCTET STRING

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

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

4.2.5 Key Usage Restriction

The keyUsageRestriction extension defines mandatory restrictions on the use of the key contained in the certificate based on policy and/or usage (e.g., signature, encryption). This field should be used whenever the use of the key is to be restricted based on either usage or policy (see discussion in policies). The usage restriction would 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).

```
keyUsageRestriction  ::=  SEQUENCE  {
     certPolicySet            SEQUENCE OF CertPolicyId OPTIONAL,
     restrictedKeyUsage       KeyUsage OPTIONAL  }
```

4.2.6 Basic Constraints

The basicConstraints extension identifies whether the subject of the certificate is a CA or an end user. In addition, this field can limit the authority of the CA in terms of the certificates it can issue. Discussion of certification path restriction is covered elsewhere in this draft. The subject type field should be present in all Internet certificates.

```
basicConstraints  ::=  SEQUENCE  {
     subjectType         SubjectType,
     pathLenConstraint       INTEGER OPTIONAL,
     permittedSubtrees       SEQUENCE OF GeneralName OPTIONAL,
     excludedSubtrees        SEQUENCE OF GeneralName OPTIONAL  }

SubjectType  ::=  BIT STRING  {
     cA                  (0),
     endEntity           (1)  }
```

4.2.7 CRL Distribution Points

The cRLDistributionPoints extension identifies the CRL distribution point or points to which a certificate user should refer to acertain if the certificate has been revoked. This extenstion provides a mechanism to divide the CRL inot manageable pieces if the CA has a large constituency.

<< Need a section which discusses the alternatives. Should permit URLs as one method to name the location for the most recent CRL. >>

4.2.8 Information Access

The informationAccess field is proposed as a private extension to tell how information about a subject or CA (or ancillary CA services) may be accessed. For example, this field might provide a pointer to information about a user (e.g., a URL) or might tell how to access CA information such as certificate status or on-line validation services. The structure of this extension is TBD.

<< Suggestions on the ASN.1 syntax are welcome. >>

4.2.9 Other extensions

The DAM defines additional extensions; however, this draft does not include them as there use is not part of the basic Internet profile.

4.3 Examples

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

5 CRL and CRL Extensions Profile

As described above, one goal of this draft is to create a profile for X.509 v2 CRLs that will foster interoperability and a reusable public key infrastructure. To achieve this goal, some assumptions need to be made about the nature of information to be included along with guidelines for how extensibility will be employed.

CRLs 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 draft is to establish a common baseline for generic applications requiring broad interoperability and limited special purpose requirements. Emphasis will be on support for X.509 v2 CRLs. The draft will define a baseline set of information along with common locations within a CRL and common representations for common information. Environments with additional requirements may build on this profile or may replace it.

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  ::=  SIGNED SEQUENCE  {
     version              Version DEFAULT v1,
     signature            AlgorithmIdentifier,
     issuer               Name,
     lastUpdate           UTCTime,
     nextUpdate           UTCTime,
     revokedCertificates  SIGNED SEQUENCE OF SEQUENCE  {
        signature            AlgorithmIdentifier,
        issuer               Name,
        userCertificate      SerialNumber,
        revocationDate       UTCTime,
        crlEntryExtensions   Extensions OPTIONAL  }  OPTIONAL,
     crlExtensions   [0]  Extensions OPTIONAL   }  }
```

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

SerialNumber  ::=  INTEGER
```

The following items describe a proposed use of the X.509 v2 CRL for the Internet.

5.1.1 Version

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

5.1.2 Signature

This field contains the algorithm identifier for the algorithm used to sign the CRL.

5.1.3 Issuer Name

The issuer name provides a globally unique identifier of the certification authority signing the CRL. The syntax of the issuer name is an X.500 distinguished name. This strawman suggests that the issuer name must provide a globally unique identifier. In addition, it should contain semantic information identifying the certification authority.

<< Any changes to 4.1.4 must be reflected here too. >>

5.1.4 Last Update

This field indicates the date on which this CRL was issued.

5.1.5 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.

5.1.6 Revoked Certificates

Revoked certificates are listed. The certificates are named by the combination of the issuer name and the user certificate serial number. The date on which the revocation occured is specified. Each revocation entry is individually signed. This profile mandates the use of same signature algorithm to sign each CRL entry and the whole CRL. CRL entry extensions are discussed in section 5.3.

5.2 CRL Extensions

The extensions already defined by ANSI X9 and ISO for X.509 v2 CRLs 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. However, an unrecognized non-critical extension may be ignored. The following presents recommended extensions used within Internet CRLs and standard locations for information. Communities may elect to use additional extensions; however, caution should be exercised in adopting any critical extensions in CRLs which might be used in a general context.

5.2.1 Authority Key Identifier

The authorityKeyIdentifier is a non-critical CRL extension that allows the CA to include an identifier of the key used to sign the CRL. This extension is useful when a CA uses more than one key. See section 7 for a discussion key changeover.

```
AuthorityKeyId  ::=  SEQUENCE  {
     keyIdentifier    [0] KeyIdentifier OPTIONAL,
     certIssuer       [1] Name OPTIONAL,
     certSerialNumber [2] CertificateSerialNumber OPTIONAL  }
     ( CONSTRAINED BY {
     -- certIssuer and certSerialNumber constitute a logical pair,
     -- and if either is present both must be present.  Either this
     -- pair or the keyIdentifier field or all shall be present. -- } )
```

5.2.2 Issuer Alternative Name

The issuerAltName is a non-critical CRL extension that provides a CA name, in a form other than an X.500 distinguished name. The syntax for the issuerAltName is the same as described in section 4.2.1. Each of the alternate names is an IA5 string. Multiple instances may be included. Whenever such alternative names are included in a CRL, the issuer alternative name field shall be used.

5.2.3 CRL Number

The cRLNumber is a non-critical CRL extension which conveys a monotonically increacing sequence number for each CRL issued by a given CA through a given CA X.500 Directory entry or CRL distribution point. This extension allows users to easily determine is a particular CRL superceeds another CRL. Use of this CRL extension is strongly encouraged.

```
CRLNumber  ::=  INTEGER
```

5.2.4 Issuing Distribution Point

The issuingDistributionPoint is a critical CRL extension that identifiers the CRL distribution point for this particular CRL, and it indicates whether the CRL covers revocation for end entities certificate only, CA certificates only, or a limitied set of reason codes. Support for CRL distribution points is strongly encouraged. However, the use of certificateHold is strongly discouraged.

```
DistributionPoint  ::=  SEQUENCE  {
     distributionPoint     DistributionPointName,
     reasons               ReasonFlags OPTIONAL  }

DistributionPointName  ::=  CHOICE  {
     fullName           [0] Name,
     nameRelativeToCA   [1] RelativeDistinguishedName  }

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

5.2.5 Delta CRL Indicator

The deltaCRLIndicator is a critical CRL extension that identifies a delta-CRL. The use of delta-CRLs is strongly discouraged. Rather, CAs are encouraged to always issue complete CRLs.

5.3 CRL Entry Extensions

The CRL entry extensions already defined by ANSI X9 and ISO for X.509 v2 CRLs 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 an critical CRL entry extension. 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.

5.3.1 Reason Code

The reasonCode is a non-critical CRL entry extension that identifies the reason for the certificate revocation. The inclusion of reason codes is encouraged. The reasonCode extension permits certificates to placed on hold or suspended. The processing associated with suspended certificates greatly complicates certificate validation. The use of this feature is strongly discouraged.

```
CRLReason  ::=  ENUMERATED  {
     unspecified             (0),
     keyCompromise           (1),
     caCompromise            (2),
     affiliationChanged      (3),
     superseded              (4),
     cessationOfOperation    (5),
     certificateHold         (6),
     certHoldRelease         (7),
     removeFromCRL           (8)  }
```

5.3.2 Expiration Date

The expirationDate is a non-critical CRL entry extension that indicates the expiration of a hold entry in a CRL. The use of this extension is strongly discouraged.

5.3.3 Instruction Code

The instructionCode 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. The use of this extension is strongly discouraged.

5.3.4 Invalidity Date

The invalidityDate is a non-critical CRL entry extension that provides the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid. This date may be earlier than the revocation date in the CRL entry (which is the date that the CA revoked the certificate). The use of this extension is encouraged.

```
InvalidityDate  ::=  GeneralizedTime
```

5.4 Examples

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

6 Certificate and CRL Distribution

6.1 Distribution via X.500

Within an X.500 Directory, the certificate for an end entity can be found in the userCertificate attribute. This attribute is normally associated with the strongAuthenticationUser object class.

Within an X.500 Directory, the certificate for a certification authority can be found in the cACertificate attribute, and the most recent CRL can be found in the certificateRevocationList attribute. These attributes are normally associated with the certificationAuthority object class.

6.2 Distribution via Electronic Mail

RFC 1424 specifies methods for key certification, certificate revocation list (CRL) storage, and CRL retrieval. These services are required of an RFC 1422 certification authority. Each service involves an electronic mail request and an electronic mail reply.

<< Need to define a format for one user to send his certificate to another. This format could be used to obtain arbitrary certificates from a certificate server or to solicit certificates from the user themselves. >>

6.3 Distribution via HTTP

<< Need to define a convention for using HTTP to obtain certificates from a server. >>

As discussed in section 4.2.7, the user certificate may contain a URL that specifies the location where the most recent CRL which could contain an entry revoking the certificate can be found. HTTP can be used to fetch the most recent CRL from this location.

6.4 On-line Certificate Validation

As discussed above, consumers of certificates must be able to determine the validity of a certificate when using the certificate. There are many possible approaches to informing consumers on the status of the certificate and these approaches have different operational characteristics. One alternative is to provide an on-

line validation service. Such a service reduces the complexity of the client applications (by moving it to the on-line service), and it provides the most timely status possible.

In addition, on-line validation servers can also help to resolve the root key management a distribution problem by providing a single trusted agent for asserting root key status where the agent is independent of the certification hierarchy itself.

The on-line validation could be performed by either the CA who issued the certificate (directly or via a delegatee) or as a general service by a "trusted" third party. Note, this service could also be extended to the validation of any certificate like item (e.g., PGP credential, DNS record, STT credential) and could facilitate application interaction between users using different certificate formats.

The general model involves a request/response format which might be transferred using a number of alternative transport protocols. In general, the requestor sends the certificate (or a user reference) along with an indication of the service to be provided. This service might be coupled with the general certificate distribution service by adding service flags to that request as well.

The request should contain: << this section is still in progress >>

```
     Certificate (or cert path)
     Service parameters
     Parse cert for me
     Check CRLs
     Result format (ASN, Text, HTML, ....)
     Sign result (with a specified algorithm)
     Other qualifiers
     Desired domain/policy OID (does this validate to a specific Root)
```

A possible Syntax:

```
ValidationRequest  ::=  SEQUENCE  {
     CertPathType    OBJECT IDENTIFIER,
     CertPath        SEQUENCE OF OCTET STRING,
     TargetRootID    Name OPTIONAL,
     ServiceParams   SEQUENCE OF ServiceParam OPTIONAL  }

ServiceParam  ::=  INTEGER  {
     ASNresult       (1),
     Textresult      (2),
     HTMLresult      (3),
     ....                  }
```

The response should contain:

Status: current/valid, expired (date), revoked (date/reason),
suspended
Cert path problem: what failed, where, and why
Policy/attribute/constraints from validated cert path
Parsed data: name, key, attributes
Could be signed by validator or rely on secure channel

A possible syntax:

```
   ValidationResponse  ::=  OPTIONALLY SIGNED SEQUENCE  {
        validator       Name OPTIONAL,
        certInfo        CHOICE  {
            cert        OCTET STRING,
            reference   IssuerSerial,
            certdata    T61 STRING  },
                         -- text including name, key, and attributes
        status          StatusCode,
        detail          ANY Defined By StatusCode OPTIONAL,
        validationData  ???? }
                        -- problems with cert path, policy attributes, etc.

   StatusCode  ::=  INTEGER  {
        valid          (1),
        revoked        (2),
        expired        (3),
        suspended      (4)  }
```

7 Key Pair Updating Procedures

A fundamental principle of the PKI is that it must be possible to update all of the cryptographic keys used, both by end entity's and by PKI components (e.g., CAs). Furthermore, for the PKI to be usable, the update of one key pair must not force the update of any other key pair or Certificate. In this section, we deal with the update of CA key pairs. Key updating for end entities is dealt with in section 9.4.

For CA key pair updating we will fulfil the following requirements:

(a) All certificates valid before the update must remain valid.

(b) A subject whose certificate is verifiable using the new CA public key must also be able to verify certificates verifiable using the old public key.

(c) End entities who directly trust the old CA key pair must be able to verify certificates signed using the new key CA private key. This

is required for situations where the old CA public key is "hardwired" into the end entity's cryptographic equipment (e.g., smartcard memory).

(d) All entities (not just those certified by that CA) must have both the new and old CA public keys available from the time of the change (whether or not they trust it is a local matter).

The basis of the scheme described below is that the CA protects its new public key using its previous private key and vice-versa. Thus when a CA updates its key pair it must generate two new cACertificate attribute values if certificates are made available using an X.500 directory.

Note that the scheme below does not make use of any of the X.509 v3 certificate extensions as it must be able to work for X.509 v1 certificates. However, the presence of the KeyIdentifier extension permits efficiency improvements.

Note that the change of a CA key affects both certificate verification and CRL checking.

It is worth noting that the operation involved here is key update, only the key pair (and related attributes) of the CA are changed. Thus, this operation cannot be used in the event of a CA key compromise.

While the scheme could be generalised to cover cases where the CA updates its key pair more than once during the validity period of one of its end entity's certificates, this generalisation seems of dubious value. Therefore, the validity period of a CA key must be greater than the validity period of any certificate issued by that CA.

We first present the data structures required then specify the steps involved in changing the CA key and the various possibilities for certificate verification. Note that the description below assumes that X.500 is used for publishing certificates. This assumption is simply for clarity of presentation, if the same data structures are published some other way, the scheme still works.

7.1 ASN.1 Data Types

```
-- existing CA cert from X.509
-- this contains the current and old CA certificate(s)
-- all entities under this CA need a local copy of
-- one of these
CACertificate  ::=  ATTRIBUTE
```

```
                           WITH ATTRIBUTE-SYNTAX Certificate

   -- Securing the old CA public key with the new private key and
   -- vice-versa. Securing the new CA public key with the old private
   -- key is needed to avoid having to issue the new CA public key
   -- using out-of-band means to entities certified using the old CA
   -- key; with this they can verify certificates signed using the new
   -- CA private key.
   -- The data structures can be stored in this X.500 attribute
   CALinkages  ::=  ATTRIBUTE
                           WITH ATTRIBUTE-SYNTAX CALinkage

   CALinkage  ::=  SIGNED SEQUENCE  {
        protectedCACertSerial     INTEGER,
                                  -- the serial number in the CACertificate
                                  -- value which we wish to link to
        protectingCACertSerial    INTEGER,
                                  -- the serial number in the CACertificate
                                  -- value which contains the public key
                                  -- corresponding to the private key used
                                  -- to sign this
        caName                    Name,
        link                      HASH Certificate  }
```

7.2 CA Operator Actions

To change the key of the CA, the CA operator does the following:

(1) Generates a new key pair.

(2) Calculate the certificate for the new key pair.

(3) Create a CALinkage (based on the old CA certificate) using the new private key.

(4) Create a CALinkage (based on the new CA certificate) using the old private key.

(5) Publish these new data structures.

7.3 Verifying Certificates

Normally when verifying a signature, the verifier simply verifies the certificate containing the public key of the signer. However, once a CA is allowed to update it's key there are a range of new possibilities. These are shown in the table below.

The term PSE (personal security environment) is used to denote

locally held and trusted information. This can only be assumed to include a single CA public key.

	CACertificate contains NEW public key		CACertificate contains only OLD public key	
	PSE Contains NEW public key	PSE Contains OLD public key	PSE Contains NEW public key	PSE Contains OLD public key
Signer's cert is protected using NEW public key	Case 1: This is the standard case where the verifier can directly verify the certificate without using the directory	Case 3: In this case the verifier must access the directory in order to get the value of the NEW public key	Case 5: Although the CA operator has not updated the directory the verifier can verify the certificate directly - this is thus the same as case 1.	Case 7: In this case the CA operator has not updated the directory attributes and so the verification will FAIL
Signer's cert is protected using OLD public key	Case 2: In this case the verifier must access the directory in order to get the value of the OLD public key	Case 4: In this case the verifier can directly verify the certificate without using the directory	Case 6: The verifier thinks this is the situation of case 2 and will access the directory, however the verification will FAIL	Case 8: Although the CA operator has not updated the directory, the verifier can verify the certificate directly -- this is thus the same as case 4.

7.3.1 Verification in cases 1, 4, 5 and 8

In these cases the verifier has a local copy of the CA public key which can be used to verify the certificate directly. This is the same as the situation where no key change has ever occurred.

Note that case 8 may arise between the time when the CA operator has generated the new key pair and the time when the CA operator stores the updated attributes in the Directory. Case 5 can only arise if the CA operator has issued both the signer's and verifier's

certificates during this "gap" (the CA operator should avoid this as it leads to the failure cases described below).

7.3.2 Verification in case 2

In case 2 the verifier must get access to the old public key of the CA. The verifier does the following:

(1) Lookup the CACertificate attribute in the directory and pick the appropriate value.

(2) Lookup the associated CALinkages attribute value.

(3) Verify that these are correct using the new CA key (which the verifier has locally).

(4) If correct then check the signer's certificate using the old CA key.

Case 2 will arise when the CA operator has issued the signer's certificate, then changed key and then issued the verifier's certificate, so it is quite a typical case.

7.3.3 Verification in case 3

In case 3 the verifier must get access to the new public key of the CA. The verifier does the following:

(1) Lookup the CACertificate attribute in the directory and pick the appropriate value.

(2) Lookup the associated CALinkages attribute value.

(3) Verify that these are correct using the old CA key (which the verifier has stored locally).

(4) If correct then check the signer's certificate using the new CA key.

Case 3 will arise when the CA operator has issued the verifier's certificate, then changed key and then issued the signer's certificate, so it is also quite a typical case.

7.3.4 Failure of verification in case 6

In this case, the CA has issued the verifier's PSE containing the new key without updating the directory attributes. This means that the verifier has no means to get a trustworthy version of the CA's old

key and so verification fails.

Note that the failure is the CA operator's fault.

7.3.5 Failure of verification in case 7

In this case the CA has issued the signer's certificate protected with the new key without updating the directory attributes. This means that the verifier has no means to get a trustworthy version of the CA's new key and so verification fails.

Note that the failure is the CA operator's fault.

7.4 Revocation - Change of CA Key

As we saw above, the verification of a certificate becomes more complex once the CA is allowed to change its key. This is also true for revocation checks as the CA may have signed the CRL using a newer private key than the one within the user's PSE. The analysis of the alternatives is exactly as for certificate verification.

8 Guidelines for Certificate Policy Definition

<< To Be Decided >>

9 Supporting Management Protocols

The certificate management protocol exchanges defined in this section support management communications between client systems, each of which supports one or more users, and CAs. In addition, one management protocol exchange is defined for use between two CAs, for the purpose of establishing cross-certificates. Each exchange is defined in terms of a sequence of messages between the two systems concerned. This section defines the contents of the messages exchanged.

The protocols for conveying these exchanges in different environments (on-line, E-mail, and WWW) are specified in Section 10.

The protocol exchanges defined in this document are:

- One-Step Registration/Certification
- User Registration
- User Initialization/Certification with Client-Generated Encryption Key Pair
- User Initialization/Certification with Centrally-Generated Encryption Key Pair
- Encryption Key Pair Recovery

- Key Pair Update for Client-Generated Key Pair
- Key Pair Update for Centrally-Generated Key Pair
- Key Pair Update (Centrally-Initiated)
- Revocation Request
- Cross-Certification

The following notes apply to the protocol exchange descriptions:

- In exchanges between a client system and a CA, the protocol exchange is initiated by the client system. The one exception to this is the Key Pair Update (Centrally-Initiated) exchange.
- To provide an upgrade path, a protocol version indicator is always included in the first message of an exchange.
- A message type indicator is included in the protected part of all messages.
- All messages include an optional transaction identifier which is used to assist correlation of request and response messages for one transaction. This identifier is generated by the initiator of the exchange and will typically include the initiator's name plus a transaction sequence number.
- The initial message from the client to the CA may optionally contain the client system time. This is used to facilitate the correction of client time problems by central administrators.
- Responses from CA to client include the CA system time. The client can use this time to check that its own system time is within a reasonable range.
- Random numbers are used in some of the protocols to prevent replay of the exchanges.
- Responses can be aborted at any time. An enumerated error code is sent from the aborting end and can be decoded into a user readable error string at the other end. Error codes are not specified in this version of this document.
- Items in square brackets [] are optional.
- In every instance in which a public key is transferred, it is transferred in the form of X.509 subjectPublicKeyInfo, including algorithm identifier and (optional) parameters.
- When a new key pair is generated by a client, a key identifier may optionally be sent to the CA along with the public key for inclusion in the certificate. However, the CA may override this value with a key identifier of its own. If the client is concerned about the key identifier value used, it should check the new certificate.
- Where this description refers to an encryption key pair, this could be a key pair for RSA key transport or could be key pair for key establishment using, for example, a Diffie-Hellman based algorithm.

Note that in this version of this document, the message contents are defined at an outline level only. A future version of this document will fill out the full details of message syntax in ASN.1.

9.1 One-Step Registration/Certification

9.1.1 Overview of Exchange

This protocol exchange is used to support registration of a user, together with request and issue of certificate(s), for use in environments in which client systems generate their own key pair or pairs. It is a simple exchange, designed

for easy implementation, but lacks some of the features and protective measures inherent in the exchanges defined subsequently. The user must have a pre-established digital signature key pair. Furthermore, the user must have a preestablished reliably-known copy of the public key of the CA concerned (this generally requires some form of off-line data exchange to ensure that the correct public key is known).

If the request is accepted by the CA, it results in the generation of certificate(s) for client-generated digital signature and/or encryption public keys.

9.1.2 Detailed Description

A single message is used for a user to register with a CA and request certificate issuance.

```
RegCertRequest:: client-to-CA
{
 protocol version
 message type
 [transaction identifier]
 [client system time]
 user unique name (DN)
 [user signature public key]
 [user signature key identifier]
 [client-generated encryption public key]
 [client-generated encryption key identifier]
 user attributes
 [certificate policy]
} Signature (signed with user signature private key)
```

No specific message is defined to return the generated certificate(s). It is assumed that the client will obtain a copy of the certificate(s) by other means and, by checking the certificate contents and CA signature, ensure that the request was processed by the correct CA.

9.2 User Registration

9.2.1 Overview of Exchange

This protocol exchange is used for a user to request registration with a CA. It is a first step in the establishment of key materials and certificates between client and CA for that user. Assuming the CA accepts the request, it will be necessary to follow-up this exchange with a User Initialization/Certification exchange as described in 9.3 or 9.4. At the time this request is issued, it is not necessary for the client to have any established key materials.

9.2.2 Detailed Description

A single message is used for a user to request registration with a CA.

```
RegisterUserRequest:: client-to-CA
{
 protocol version
 message type
 [transaction identifier]
 [client system time]
 user unique name (DN)
 user attributes
 [certificate policy]
} Signature (signed with user signature private key)
```

No specific message is defined to respond to this request. It is asumed that the procedure defined in 9.3 or 9.4 will follow.

9.3 User Initialization/Certification with Client-Generated Encryption Key Pair

9.3.1 Overview of Exchange

This protocol exchange is used to support client initialization, including certificate issuance, for one user, with provision for simultaneously establishing and certifying separate key pairs for digital signature and encryption (or encryption key exchange) purposes. Both key pairs are generated by the client and no private key is exposed to the CA. Generation and certification of the encryption key pair is optional.

Prior to conducting this exchange, the user must have registered with the CA, either using the user registration exchange defined in 9.2 or by other means.

Following registration, the CA creates a secret data item, called an authorization code, and transfers this data item by out-of-band means to the user. The authorization code is used to establish authentication and integrity protection of the user initialization/certification on-line exchange. This is done by generating a symmetric key based on the authorization code and using this symmetric key for generating Message Authentication Codes (MACs) on all exchanges between client and CA.

In the first two messages exchanged, the client sends its user signature public key (and, optionally, a client-generated encryption public key) to the CA and the CA returns the currently valid CA certificate(s). This exchange of public keys allows the client and CA to authenticate each other.

9.3.2 Detailed Description

The user receives a reference number and a secret machine-generated authorization code from the CA administrator. Both pieces of information are transferred to the user in a secure manner which preserves their integrity and confidentiality. The reference number is used to uniquely identify the client at the CA and the authorization code is used to secure the exchange integrity-wise. The reference number is used instead of a DN to uniquely identify the client because a DN may be lengthy and difficult for a user to manually type without error.

After the reference number and authorization code have been entered by the user, the client generates:

- a client random number,
- (if a new user signature key pair is required) a new user signature key pair,
- (if a new client-generated encryption key pair is required) a new encryption key pair.

The client securely stores locally any new signature private key and/or client-generated encryption private key. The client then sends the message InitClientRequest to the CA. The entire structure is protected from modification with a MAC based on the authorization code.

```
InitClientRequest:: client-to-CA
{
 protocol version
 message type
 [transaction identifier]
 [client system time]
```

```
 client random number
 reference number
 user signature public key
 [user signature key id]
 [client-generated encryption public key]
 [client-generated encryption key id]
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the InitClientRequest structure, if the CA recognizes the reference number and if the protocol version is valid, it saves the client random number, generates its own random number (CA random number), and validates the MAC. Then for the user encryption public key, it creates:

- a new certificate for the user?s digital signature public key,
- (if a new client-generated encryption key pair is required) a new certificate.

The CA responds to the client with the message InitClientResponse. The entire structure is protected from modification with a MAC based on the authorization code.

```
InitClientResponse:: CA-to-client
{
 message type
 [transaction identifier]
 client random number
 CA random number
 CA signature public key certificate
 new user signature public-key certificate
 [new user encryption public-key certificate]
 CA system time
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the InitClientResponse structure, the client checks that its own system time is sufficiently close to the CA system time, checks the client random number, and validates the MAC. The client then securely stores the new certificates and acknowledges the transaction by sending back the message InitClientConfirm. The fields in this message are protected from modification with a MAC based on the authorization code.

```
InitClientConfirm:: client-to-CA
{
 message type
 [transaction identifier]
```

```
    client random number
    CA random number
    MAC algorithm id
   } MAC (key based on authorization code)
```

Upon receipt of the InitClientConfirm structure, the CA checks the random numbers and validates the MAC. If no errors occur, the CA archives the new user public-key certificate(s).

9.4 User Initialization/Certification with Centrally-Generated Encryption Key Pair

9.4.1 Overview of Exchange

This protocol exchange is used to support client initialization, including certificate issuance, for one user, with provision for simultaneously establishing and certifying separate key pairs for digital signature and encryption (or encryption key exchange) purposes. The digital signature key pair is generated by the client. Optionally, a new encryption key pair is generated by (and, optionally, backed up by) a central facility associated with the CA.

Prior to conducting this exchange, the user must have registered with the CA, either using the user registration exchange defined in 9.2 or by other means.

Following registration, the CA creates a secret data item, called an authorization code, and transfers this data item by out-of-band means to the user. The authorization code is used to establish authentication and integrity protection of the user initialization/certification on-line exchange. This is done by generating a symmetric key based on the authorization code and using this symmetric key for generating Message Authentication Codes (MACs) on all exchanges between client and CA.

In the first two messages exchanged, the client sends its user signature public key to the CA and the CA returns the currently valid CA certificate(s). This exchange of public keys allows the client and CA to authenticate each other.

If a centrally-generated encryption key pair is to be established, the private key of the newly generated key pair is sent from the CA to the client. The client first generates a protocol encryption key pair and sends the public protocol encryption key to the CA. The CA creates a random symmetric key called the session key and encrypts the user encryption private key with it and then encrypts the session key with the public protocol encryption key it received from the client. The CA sends the encrypted user encryption private key and

encrypted session key back to the client. The client uses its private protocol decryption key to decrypt the session key and then uses the session key to decrypt the encryption private key. The protocol encryption key pair and session key are discarded after the exchange.

9.4.2 Detailed Description

The user receives a reference number and a secret machine-generated authorization code from the CA administrator. Both pieces of information are transferred to the user in a secure manner which preserves their integrity and confidentiality. The reference number is used to uniquely identify the client at the CA and the authorization code is used to secure the exchange integrity-wise. The reference number is used instead of a DN to uniquely identify the client because a DN may be lengthy and difficult for a user to manually type without error.

After the reference number and authorization code have been entered by the user, the client generates:

- a client random number,
- (if a new user signature key pair is required) a new user signature key pair,
- (if a new centrally-generated encryption key pair is required) a protocol encryption key pair.

The client securely stores locally any new signature private key and/or client-generated encryption private key. The client then sends the message InitCentralRequest to the CA. The entire structure is protected from modification with a MAC based on the authorization code.

```
InitCentralRequest:: client-to-CA
{
 protocol version
 message type
 [transaction identifier]
 [client system time]
 client random number
 reference number
 user signature public key
 [user signature key id]
 [protocol encryption key]
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the InitCentralRequest structure, if the CA recognizes the reference number and if the protocol version is valid,

it saves the client random number, generates its own random number (CA random number), and validates the MAC. It then creates:

- a new certificate for the user?s digital signature public key,
- (if a new centrally-generated encryption key pair is required) a session key, a new user encryption key pair, and a new certificate for the user encryption public key.

The CA responds to the client with the message InitCentralResponse. If a new centrally-generated encryption key pair is being generated, the user encryption private key is encrypted using the session key and the session key is encrypted with the protocol encryption public key. The entire structure is protected from modification with a MAC based on the authorization code.

```
InitCentralResponse:: CA-to-client
{
 message type
 [transaction identifier]
 client random number
 CA random number
 CA signature public key certificate
 new user signature public-key certificate
 [new user encryption public-key certificate]
 [new user encryption private key encrypted with session key]
 [session key encrypted with protocol encryption key]
 CA system time
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the InitCentralResponse structure, the client checks that its own system time is sufficiently close to the CA system time, checks the client random number, and validates the MAC. If a new centrally-generated encryption key pair is included, the client decrypts the encryption private key. The client then securely stores the new certificates and encryption private key (if present) and acknowledges the transaction by sending back the message InitCentralConfirm. The fields in this message are protected from modification with a MAC based on the authorization code.

```
InitCentralConfirm:: client-to-CA
{
 message type
 [transaction identifier]
 client random number
 CA random number
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the InitCentralConfirm structure, the CA checks the random numbers and validates the MAC. If no errors occur, the CA archives the new user public-key certificate(s) and (if there is a new centrally-generated encryption key pair and key recovery is to be supported) the encryption private key.

9.5 Encryption Key-Pair Recovery

9.5.1 Overview of Exchange

This protocol exchange is used to support recovery in the event that a client no longer has a valid signature key pair (due to expiration or revocation), or client system key materials have been lost, e.g., as a result of a forgotten user password. This exchange assumes a system in which an encryption key pair has been centrally generated and backed up (by a central system associated with a CA).

This exchange is very similar to the exchange for User Initialization/Certification with Centrally-Generated Encryption Key Pair. The client and CA start without a way to trust one another, i.e., they have no reliable shared key pairs.

9.5.2 Detailed Description

The user must first receive, by out-of-band means, a reference number and a secret machine-generated authorization code from the CA administrator. The on-line exchange then consists of a sequence of KeyRecoverRequest, KeyRecoverResponse and KeyRecoverConfirm, which are the same as the exchange in 9.4 except for two differences. First, the CA does not generate (or archive) a new encryption key pair and encryption public-key certificate for the user. Second, the user?s entire encryption key history (list of encryption public-key certificates and matching encryption private keys) are sent back to the client with KeyRecoverResponse.

```
KeyRecoverRequest:: client-to-CA
{
 protocol version
 message type
 [transaction identifier]
 [client system time]
 client random number
 reference number
 user signature public key
 [user signature key id]
 protocol encryption key
 MAC algorithm id
} MAC (key based on authorization code)
```

```
KeyRecoverResponse:: CA-to-client
{
 message type
 [transaction identifier]
 client random number
 CA random number
 CA certificate(s)
 user encryption private key history encrypted with session key
 session key encrypted with protocol encryption key
  user encryption public-key certificate history
 new user signature public-key certificate
 CA system time
 MAC algorithm id
} MAC(key based on authorization code)

KeyRecoverConfirm:: client-to-CA
{
 message type
 [transaction identifier]
 client random number
 CA random number
 MAC algorithm id
} MAC (key based on authorization code)
```

9.6 Key Pair Update for Client-Generated Key Pair(s)

9.6.1 Overview of Exchange

This exchange is used to update the signature key pair and/or client-generated encrypyion key pair of a user, (e.g., as a result of routine cryptoperiod expiry).

A user must have a valid signature key pair in order to do this exchange. It is up to the client to determine when a new signature key pair should be generated; this has to be done prior to the expiration of its signature public-key certificate.

A key pair update request from a client is digitally signed using the original user signature private key, this signature being verifiable using an existing signature certificate. If the key pair update is for a new user digital signature key, then the client signs the request message once more (including the first signature), this time using the new signature private key. The reason for this second signature is to prove to the CA that the client possesses both the new and old private keys.

The request is verified at the CA by using the matching user signature public key. A protocol signature key pair is used to

authenticate messages from the CA to the client. CA responses are signed with the protocol signature private key.

A CA response is validated at the client by using a protocol signature public-key certificate which is included in the CA response. The protocol signature public-key certificate can be validated by using the CA certificate stored at the client. A new user initialization (as in 9.3) or key pair recovery (as in 9.4) must be done if the user signature key pair becomes invalid.

In some client system implementations, local key materials are stored in an encrypted key data disk file. A user may have several copies of this key data file on different computers. It is possible that a key update could occur and the user could forget to copy the updated key data file to all the computers they use. To help keep the client using the latest keys, the client sends the CA the serial number of the latest user signature public-key certificate it has in the key update request. Serial numbers are sent so that the CA can check if the client has the latest key pair. If the client does not have the latest signature private key and the signature public-key certificate serial number is equal to that of a previous certificate, the CA sends back an error code which indicates that the client has an old version of the key data file. After this, the client can either find the latest key data file or, if that fails, key recovery can be done.

9.6.2 Detailed Description

The client initiates the exchange by creating a new signature and/or encryption key pair and generating a random number (client random number). The client then sends the CA the message UpdateClientKeyRequest. The fields in this message are protected from modification and authenticated by a digital signature using the pre-existing user signature private key. If the update includes a new signature key pair, the result is additionally signed using the new user signature private key.

```
UpdateClientKeyRequest:: client-to-CA
{
 protocol version
 message type
 [transaction identifier]
 [client system time]
 client random number
 user unique name (DN)
 [new user signature public key]
 [new user signature key id]
 [new user encryption public key]
 [new user encryption key id]
```

```
 serial number of latest signature public-key certificate
} Signature (signed with pre-existing user signature private key)
[Signature (signed with new user signature private key)]
```

Upon receipt of the UpdateClientKeyRequest structure, the CA checks the protocol version, checks the serial number, saves the client random number, generates its own random number (CA random number) and verifies the signature using the previous user verification key which is archived at the CA. If a user digital signature key pair is being updated, the CA also checks the second signature. It then generates new user signature and/or encryption public-key certificate(s). The CA responds with the message UpdateClientKeyResponse. The fields in this message are protected from modification and authenticated by a digital signature using the CA protocol signature private key.

```
UpdateClientKeyResponse:: CA-to-client
{
 message type
 [transaction identifier]
 client random number
 CA random number
 protocol signature public-key certificate
 [new user signature public-key certificate]
 [new user encryption public-key certificate]
 CA system time
} Signature (signed with protocol signature private key)
```

Upon receipt of the UpdateClientKeyResponse structure, the client verifies the digital signature using the protocol verification key contained in the protocol signature public-key certificate, checks that its own system time is close to the CA system time, and checks the received client random number. The client then securely stores locally the new user public-key certificate(s). It responds with the message UpdateClientKeyConfirm. The fields in this message are protected from modification and authenticated by a digital signature using the pre-existing user signature private key.

```
UpdateClientKeyConfirm:: client-to-CA
{
 message type
 [transaction identifier]
 client random number
 CA random number
} Signature (signed with pre-existing user signature private key)
```

Upon receipt of the UpdateClientKeyConfirm structure, the CA checks that the client and CA random numbers are the same as the ones initially generated, and verifies the received signature using the

previous user signature public key which is archived at the CA. The CA then archives the new user public-key certificate(s) and updates its data stores to reflect the new status of the user.

9.7 Key Pair Update for Centrally-Generated Encryption Key Pair

9.7.1 Overview of Exchange

This exchange is used to update the encryption key pair of an user, under the assumption that encryption key pairs are generated (and, optionally, backed up) centrally. A user must have a valid signature key pair in order to do this exchange. It is up to the client to determine when a new encryption key pair should be generated; this has to be done some time before the expiration date in its encryption public-key certificate.

9.7.2 Detailed Description

The client initiates the exchange by generating a random number (client random number) and a protocol encryption key pair. The client then sends the CA the message UpdateEncKeyRequest1. The fields in this message are protected from modification and authenticated by a digital signature using the latest user signature private key.

```
UpdateCentralKeyRequest:: client-to-CA
{
 protocol version
message type
 [transaction identifier]
 [client system time]
 client random number
 user unique name (DN)
 latest user encryption public-key certificate serial number
 latest user signature public-key certificate serial number
 protocol encryption key
} Signature (signed with latest user signature private key)
```

Upon receipt of the UpdateCentralKeyRequest structure, the CA checks the protocol version, checks the serial numbers, saves the client random number, generates its own random number (CA random number), generates a session key, and verifies the received signature using the latest user signature public key which is archived at the CA. It then generates a new end-user encryption key pair and encryption public-key certificate for the user. In the case where the encryption public-key certificate serial number is the second latest, the CA does not generate any keys and uses the latest encryption public-key certificate and encryption private key that it has. The CA responds with the message UpdateEncKeyResponse1. In this message, the new or

latest encryption private key is encrypted with the session key and the session key is encrypted with the protocol encryption key. The fields in this message are protected from modification and authenticated by a digital signature using the protocol signature private key.

```
UpdateCentralKeyResponse:: CA-to-client
{
 message type
 [transaction identifier]
 client random number
 CA random number
 new or latest user encryption private key encrypted with session key
 new or latest user encryption public-key certificate
  session key encrypted with protocol encryption key
 protocol signature public-key certificate
 CA system time
} Signature (signed with protocol signature private key)
```

Upon receipt of the UpdateCentralKeyResponse structure, the client verifies the digital signature using the protocol signature public-key certificate, makes sure its own system time is close to the CA system time, and checks the received client random number. The client then decrypts the new or latest encryption private key and securely stores locally the new or latest user encryption public-key certificate and encryption private key. It responds with the message UpdateCentralKeyConfirm. The fields in this message are protected from modification and authenticated by a digital signature using the latest user signature private key.

```
UpdateCentralKeyConfirm:: client-to-CA
{
 message type
 [transaction identifier]
 client random number
 CA random number
} Signature (signed with latest user signature private key)
```

Upon receipt of the UpdateClientKeyConfirm structure, the CA checks that the client and CA random numbers are correct and verifies the signature using the latest user signature public key which is archived at the CA. If no errors occur, the CA archives the new user encryption public-key certificate and encryption private key, and updates its data stores to reflect the new status of the user.

9.8 Key Pair Update (Centrally-Initiated)

9.8.1 Overview of Exchange

This exchange is used to update the encryption key pair of an user, under the assumption that encryption key pairs are generated (and, optionally, backed up) centrally. This exchange differs from the preceding exchange (Key Pair Update for Centrally-Generated Encryption Key Pair) in that the exchange is initiated by the CA rather than the client.

9.8.2 Detailed Description

<< To be supplied >>

9.9 Revocation Request

9.9.1 Overview of Exchange

This protocol exchange is used to support a revocation request from a user or other authorized party.

9.9.2 Detailed Description

<< To be supplied >>

9.10 Cross-Certification

9.10.1 Overview of Exchange

The cross certification exchange allows two CAs to simultaneously certify each other. This means that each CA will create a certificate that contains the CA verification key of the other CA.

Cross certification is initiated at one CA known as the responder. The CA administrator for the responder identifies the CA it wants to cross certify and the responder CA equipment generates an authorization code. The responder CA administrator passes this authorization code by out-of-band means to the requester CA administrator. The requester CA administrator enters the authorization code at the requester CA in order to initiate the on-line exchange.

The authorization code is used for authentication and integrity purposes. This is done by generating a symmetric key based on the authorization code and using the symmetric key for generating Message Authentication Codes (MACs) on all messages exchanged.

Serial numbers and protocol version are used in the same manner as in the above CA-client exchanges.

9.10.2 Detailed Description

The requester CA initiates the exchange by generating a random number (requester random number). The requester CA then sends the responder CA the message CrossCertifyRequest. The fields in this message are protected from modification with a MAC based on the authorization code.

```
CrossCertifyRequest:: requester CA to responder CA
{
 protocol version
 message type
 [transaction identifier]
 requester random number
 requester CA unique name (DN)
 requester CA public key
 [requester CA key id]
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the CrossCertifyRequest structure, the responder CA checks the protocol version, saves the requester random number, generates its own random number (reponder random number) and validates the MAC. It then generates and archives a new requester certificate which contains the requester CA public key and is signed with the responder CA signature private key. The responder CA responds with the message CrossCertifyResponse. The fields in this message are protected from modification with a MAC based on the authorization code.

```
CrossCertifyResponse:: responder CA to requester CA
{
 message type
 [transaction identifier]
 requester random number
 reponder random number
 requester certificate (requester CA is the subject, signed by responder CA)
 responder CA public key
 [responder CA key id]
 responder CA system time
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the CrossCertifyResponse structure, the requester CA checks that its own system time is close to the responder CA system

time, checks the received random numbers and validates the MAC. It then generates and archives a new responder certificate which contains the responder CA public key and is signed by the requester CA signature private key. The requester CA responds with the message CrossCertifyConfirm. The fields in this message are protected from modification with a MAC based on the authorization code.

```
CrossCertifyConfirm:: requester CA to responder CA
{
 message type
 [transaction identifier]
 requester random number
 reponder random number
 reponder certificate (responder CA is the subject, signed by requester CA)
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the CrossCertifyConfirm structure, the responder CA checks the random numbers, archives the reponder certificate, and validates the MAC. It writes both the request and reponder certificates to the Directory. It then responds with the message CrossCertifyFinish. The fields in this message are protected from modification with a MAC based on the authorization code.

```
CrossCertifyFinish:: responder CA to requester CA
{
 message type
 [transaction identifier]
 requester random number
 responder random number
 MAC algorithm id
} MAC (key based on authorization code)
```

Upon receipt of the CrossCertifyFinish message, the requester CA checks the random numbers and validates the MAC. The requester CA writes both the requester and reponder certificates to the Directory.

10 Management Protocol Transport

10.1 On-line Management Protocol

<< To be supplied. This subsection will specify a means for conveying ASN.1-encoded messages for the protocol exchanges described in Section 9 over a TCP connection. >>

10.2 Management Protocol via E-mail

<< To be supplied. This subsection will specify a means for conveying ASN.1-encoded messages for the protocol exchanges described in Section 9 via Internet mail. >>

10.3 Management Protocol via HTTP

<< To be supplied. This subsection will specify a means for conveying ASN.1-encoded messages for the protocol exchanges described in Section 9 over WWW browser-server links, employing HTTP or related WWW protocols. >>

11 Algorithm Support

11.1 One-way Hash Functions

One-way hash functions are also called message digest algorithms. MD5 and SHA-1 will be the most popular one-way hash functions used in the Internet PKI. However, PEM uses MD2 for certificates [RFC1422, RFC1423]. For this reason, MD2 will continue to be used in certificates for many years.

11.1.1 MD5 One-way Hash Function

MD5 was developed by Ron Rivest, and RSA Data Security has placed the MD5 algorithm in the public domain. MD5 is fully described in RFC 1321.

MD5 is the one-way hash function of choice for use with the RSA signature algorithm.

11.1.2 MD2 One-way Hash Function

MD2 was also 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 MD5 is preferred. MD2 is fully described in RFC 1319.

11.1.3 SHA-1 One-way Hash Function

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

SHA-1 is the one-way hash function of choice for use with the DSA signature algorithm.

11.2 Signature Algorithms

RSA and DSA will be the most popular signature algorithms used in the Internet PKI.

There is some ambiguity in 1988 X.509 document regarding the definition of the SIGNED macro regarding, the representation of a signature in a certificate or a CRL. The interpretation selected for the Internet requires that the data to be signed (e.g., the one-way function output value) is first ASN.1 encoded as an OCTET STRING and the result is encrypted (e.g., using RSAEncryption) to form the signed quantity, which is then ASN.1 encoded as a BIT STRING.

11.2.1 RSA Signature Algorithm

The RSA algorithm is named for it's inventors: Rivest, Shamir, and Adleman. The RSA signature algorithm is defined in PKCS #1. It combines the either the MD2 or the MD5 one-way hash function with the RSA asymmetric encryption algorithm. As defined in PKCS #1, the ASN.1 object identifiers used to identify these signature algorithms are:

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

md4WithRSAEncryption OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1)
    pkcs-1(1) 4  }

<< Should we permit RSA with SHA-1? >>
```

When this object identifier is used with the ASN.1 type AlgorithmIdentifier, the parameters component of that type is the ASN.1 type NULL.

11.2.2 DSA Signature Algorithm

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. The ASN.1 object identifiers used to identify this signature algorithm is:

```
dsaWithSHA-1 OBJECT IDENTIFIER  ::=  {
    joint-iso-ccitt(2) country(16) US(840) organization(1)
    us-government(101) dod(2) infosec(1) algorithms(1) 2  }
```

When this object identifier is used with the ASN.1 type AlgorithmIdentifier, the parameters component of that type is optional. If it is absent, the DSA parameters p, q, and g are assumed to be known, otherwise the parameters are included using the following ASN.1 structure:

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

Security Considerations

This entire memo is about security mechanisms.

Author Addresses:

Russell Housley
SPYRUS
PO Box 1198
Herndon, VA 22070
USA
housley@spyrus.com

Warwick Ford
Bell-Northern Research
PO Box 3511, Station C
Ottawa, Ontario
Canada KY 4H7
wford@bnr.ca

Stephen Farrell
Software and Systems Engineering Ltd
7984 Fitzwilliam Court
Dublin 2
IRELAND
stephen.farrell@sse.ie

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