Network Working Group
Request for Comments: 3279
Obsoletes: 2528
Category: Standards Track

W. Polk
NIST
R. Housley
RSA Laboratories
L. Bassham
NIST

April 2002

Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

Status of this Memo

Copyright Notice

Abstract

This document specifies algorithm identifiers and ASN.1 encoding formats for digital signatures and subject public keys used in the Internet X.509 Public Key Infrastructure (PKI). Digital signatures are used to sign certificates and certificate revocation list (CRLs). Certificates include the public key of the named subject.

Table of Contents

1 Introduction

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

This document specifies algorithm identifiers and ASN.1 [X.660] encoding formats for digital signatures and subject public keys used in the Internet X.509 Public Key Infrastructure (PKI). This specification supplements [RFC 3280], "Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile." Implementations of this specification MUST also conform to RFC 3280.

This specification defines the contents of the signatureAlgorithm, signatureValue, signature, and subjectPublicKeyInfo fields within Internet X.509 certificates and CRLs.

This document identifies one-way hash functions for use in the generation of digital signatures. These algorithms are used in conjunction with digital signature algorithms.

This specification describes the encoding of digital signatures generated with the following cryptographic algorithms:

* Rivest-Shamir-Adelman (RSA);
* Digital Signature Algorithm (DSA); and
* Elliptic Curve Digital Signature Algorithm (ECDSA).

This document specifies the contents of the subjectPublicKeyInfo field in Internet X.509 certificates. For each algorithm, the appropriate alternatives for the the keyUsage extension are provided. This specification describes encoding formats for public keys used with the following cryptographic algorithms:

* Rivest-Shamir-Adelman (RSA);
* Digital Signature Algorithm (DSA);
* Diffie-Hellman (DH);
* Key Encryption Algorithm (KEA);

* Elliptic Curve Digital Signature Algorithm (ECDSA); and
* Elliptic Curve Diffie-Hellman (ECDH).

2 Algorithm Support

This section describes cryptographic algorithms which may be used with the Internet X.509 certificate and CRL profile [RFC 3280]. This section describes one-way hash functions and digital signature algorithms which may be used to sign certificates and CRLs, and identifies object identifiers (OIDs) for public keys contained in a certificate.

Conforming CAs and applications MUST, at a minimum, support digital signatures and public keys for one of the specified algorithms. When using any of the algorithms identified in this specification, conforming CAs and applications MUST support them as described.

2.1 One-way Hash Functions

This section identifies one-way hash functions for use in the Internet X.509 PKI. One-way hash functions are also called message digest algorithms. SHA-1 is the preferred one-way hash function for the Internet X.509 PKI. However, PEM uses MD2 for certificates [RFC 1422] [RFC 1423] and MD5 is used in other legacy applications. For these reasons, MD2 and MD5 are included in this profile. The data that is hashed for certificate and CRL signing is fully described in [RFC 3280].

2.1.1 MD2 One-way Hash Function

MD2 was developed by Ron Rivest for RSA Security. RSA Security has recently placed the MD2 algorithm in the public domain. Previously, RSA Data Security had granted license for use of MD2 for non-commercial Internet Privacy-Enhanced Mail (PEM). MD2 may continue to be used with PEM certificates, but SHA-1 is preferred. MD2 produces a 128-bit "hash" of the input. MD2 is fully described in [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 one can find two different messages that 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.

2.1.2 MD5 One-way Hash Function

MD5 was developed by Ron Rivest for RSA Security. RSA Security has placed the MD5 algorithm in the public domain. MD5 produces a 128-bit "hash" of the input. MD5 is fully described in [RFC 1321].

Den Boer and Bosselaers [DB94] have found pseudo-collisions for MD5, but there are no other known cryptanalytic results. The use of MD5 for new applications is discouraged. It is still reasonable to use MD5 to verify existing signatures.

2.1.3 SHA-1 One-way Hash Function

SHA-1 was developed by the U.S. Government. SHA-1 produces a 160-bit "hash" of the input. SHA-1 is fully described in [FIPS 180-1]. RFC 3174 [RFC 3174] also describes SHA-1, and it provides an implementation of the algorithm.

2.2 Signature Algorithms

Certificates and CRLs conforming to [RFC 3280] may be signed with any public key signature algorithm. The certificate or CRL indicates the algorithm through an algorithm identifier which appears in the signatureAlgorithm field within the Certificate or CertificateList. This algorithm identifier is an OID and has optionally associated parameters. This section identifies algorithm identifiers and parameters that MUST be used in the signatureAlgorithm field in a Certificate or CertificateList.

Signature algorithms are always used in conjunction with a one-way hash function.

This section identifies OIDS for RSA, DSA, and ECDSA. The contents of the parameters component for each algorithm vary; details are provided for each algorithm.

The data to be signed (e.g., the one-way hash function output value) is formatted for the signature algorithm to be used. Then, a private key operation (e.g., RSA encryption) is performed to generate the signature value. This signature value is then ASN.1 encoded as a BIT STRING and included in the Certificate or CertificateList in the signature field.

2.2.1 RSA Signature Algorithm

The RSA algorithm is named for its inventors: Rivest, Shamir, and Adleman. This profile includes three signature algorithms based on the RSA asymmetric encryption algorithm. The signature algorithms combine RSA with either the MD2, MD5, or the SHA-1 one-way hash functions.

The signature algorithm with SHA-1 and the RSA encryption algorithm is implemented using the padding and encoding conventions described in PKCS #1 [RFC 2313]. The message digest is computed using the SHA-1 hash algorithm.

The RSA signature algorithm, as specified in PKCS #1 [RFC 2313] includes a data encoding step. In this step, the message digest and the OID for the one-way hash function used to compute the digest are combined. When performing the data encoding step, the md2, md5, and id-sha1 OIDs MUST be used to specify the MD2, MD5, and SHA-1 one-way hash functions, respectively:

```
md2  OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) US(840) rsadsi(113549)
     digestAlgorithm(2) 2 }

md5  OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) US(840) rsadsi(113549)
     digestAlgorithm(2) 5 }

id-sha1  OBJECT IDENTIFIER ::= {
     iso(1) identified-organization(3) oiw(14) secsig(3)
     algorithms(2) 26 }
```

The signature algorithm with MD2 and the RSA encryption algorithm is defined in PKCS #1 [RFC 2313]. As defined in PKCS #1 [RFC 2313], the ASN.1 OID 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 signature algorithm with MD5 and the RSA encryption algorithm is defined in PKCS #1 [RFC 2313]. As defined in PKCS #1 [RFC 2313], the ASN.1 OID used to identify this signature algorithm is:

```
md5WithRSAEncryption OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
    pkcs-1(1) 4  }
```

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

```
sha-1WithRSAEncryption OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
    pkcs-1(1) 5  }
```

When any of these three OIDs appears within the ASN.1 type AlgorithmIdentifier, the parameters component of that type SHALL be the ASN.1 type NULL.

The RSA signature generation process and the encoding of the result is described in detail in PKCS #1 [RFC 2313].

2.2.2 DSA Signature Algorithm

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

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

When the id-dsa-with-sha1 algorithm identifier appears as the algorithm field in an AlgorithmIdentifier, the encoding SHALL omit the parameters field. That is, the AlgorithmIdentifier SHALL be a SEQUENCE of one component: the OBJECT IDENTIFIER id-dsa-with-sha1.

The DSA parameters in the subjectPublicKeyInfo field of the certificate of the issuer SHALL apply to the verification of the signature.

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  }
```

2.2.3 ECDSA Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in [X9.62]. The ASN.1 object identifiers used to identify ECDSA are defined in the following arc:

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

id-ecSigType OBJECT IDENTIFIER  ::=  {
     ansi-X9-62 signatures(4) }
```

ECDSA is used in conjunction with the SHA-1 one-way hash function. The ASN.1 object identifier used to identify ECDSA with SHA-1 is:

```
ecdsa-with-SHA1  OBJECT IDENTIFIER ::= {
     id-ecSigType 1 }
```

When the ecdsa-with-SHA1 algorithm identifier appears as the algorithm field in an AlgorithmIdentifier, the encoding MUST omit the parameters field. That is, the AlgorithmIdentifier SHALL be a SEQUENCE of one component: the OBJECT IDENTIFIER ecdsa-with-SHA1.

The elliptic curve parameters in the subjectPublicKeyInfo field of the certificate of the issuer SHALL apply to the verification of the signature.

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

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

2.3 Subject Public Key Algorithms

Certificates conforming to [RFC 3280] may convey a public key for any public key algorithm. The certificate indicates the algorithm through an algorithm identifier. This algorithm identifier is an OID and optionally associated parameters.

This section identifies preferred OIDs and parameters for the RSA, DSA, Diffie-Hellman, KEA, ECDSA, and ECDH algorithms. Conforming CAs MUST use the identified OIDs when issuing certificates containing

public keys for these algorithms. Conforming applications supporting any of these algorithms MUST, at a minimum, recognize the OID identified in this section.

2.3.1 RSA Keys

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

The RSA public key MUST be encoded using the ASN.1 type RSAPublicKey:

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

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

This OID 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 [RFC 3280]). The use of a single key for both signature and encryption purposes is not recommended, but is not forbidden.

If the keyUsage extension is present in an end entity certificate which conveys an RSA public key, any combination of the following values MAY be present:

```
digitalSignature;
nonRepudiation;
keyEncipherment; and
dataEncipherment.
```

If the keyUsage extension is present in a CA or CRL issuer certificate which conveys an RSA public key, any combination of the following values MAY be present:

```
digitalSignature;
nonRepudiation;
```

keyEncipherment;
dataEncipherment;
keyCertSign; and
cRLSign.

However, this specification RECOMMENDS that if keyCertSign or cRLSign is present, both keyEncipherment and dataEncipherment SHOULD NOT be present.

2.3.2 DSA Signature Keys

The Digital Signature Algorithm (DSA) is defined in the Digital Signature Standard (DSS) [FIPS 186]. The DSA OID supported by this profile is:

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

The id-dsa algorithm syntax includes optional domain parameters. These parameters are commonly referred to as p, q, and g. When omitted, the parameters component MUST be omitted entirely. That is, the AlgorithmIdentifier MUST be a SEQUENCE of one component: the OBJECT IDENTIFIER id-dsa.

If the DSA domain parameters are present in the subjectPublicKeyInfo AlgorithmIdentifier, the parameters are included using the following ASN.1 structure:

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

The AlgorithmIdentifier within subjectPublicKeyInfo is the only place within a certificate where the parameters may be used. If the DSA algorithm parameters are omitted 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 domain parameters are omitted from the SubjectPublicKeyInfo AlgorithmIdentifier and the CA signed the subject certificate using a signature algorithm other than DSA, then the subject’s DSA domain parameters are distributed by other means. If the subjectPublicKeyInfo AlgorithmIdentifier field omits the parameters component, the CA signed the subject with a signature algorithm other than DSA, and the subject’s DSA parameters are not available through other means, then clients MUST reject the certificate.

The DSA public key MUST be ASN.1 DER 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
```

If the keyUsage extension is present in an end entity certificate which conveys a DSA public key, any combination of the following values MAY be present:

digitalSignature;
nonRepudiation;

If the keyUsage extension is present in a CA or CRL issuer certificate which conveys a DSA public key, any combination of the following values MAY be present:

digitalSignature;
nonRepudiation;
keyCertSign; and
cRLSign.

2.3.3 Diffie-Hellman Key Exchange Keys

The Diffie-Hellman OID supported by this profile is defined in [X9.42].

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

The dhpublicnumber OID 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, have the ASN.1 type DomainParameters for this algorithm.

```
DomainParameters ::= SEQUENCE {
      p       INTEGER, -- odd prime, p=jq +1
      g       INTEGER, -- generator, g
      q       INTEGER, -- factor of p-1
      j       INTEGER OPTIONAL, -- subgroup factor
      validationParms  ValidationParms OPTIONAL }

ValidationParms ::= SEQUENCE {
      seed             BIT STRING,
      pgenCounter      INTEGER }
```

The fields of type DomainParameters have the following meanings:

p identifies the prime p defining the Galois field;

g specifies the generator of the multiplicative subgroup of order g;

q specifies the prime factor of p-1;

j optionally specifies the value that satisfies the equation p=jq+1 to support the optional verification of group parameters;

seed optionally specifies the bit string parameter used as the seed for the domain parameter generation process; and

pgenCounter optionally specifies the integer value output as part of the of the domain parameter prime generation process.

If either of the domain parameter generation components (pgenCounter or seed) is provided, the other MUST be present as well.

The Diffie-Hellman public key MUST 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.

```
DHPublicKey ::= INTEGER -- public key, y = g^x mod p
```

If the keyUsage extension is present in a certificate which conveys a DH public key, the following values may be present:

keyAgreement;
encipherOnly; and
decipherOnly.

If present, the keyUsage extension MUST assert keyAgreement and MAY assert either encipherOnly and decipherOnly. The keyUsage extension MUST NOT assert both encipherOnly and decipherOnly.

2.3.4 KEA Public Keys

This section identifies the preferred OID and parameters for the inclusion of a KEA public key in a certificate. The Key Exchange Algorithm (KEA) is a key agreement algorithm. Two parties may generate a "pairwise key" if and only if they share 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-keyExchangeAlgorithm  OBJECT IDENTIFIER   ::=
       { 2 16 840 1 101 2 1 1 22 }

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

CAs MUST populate the parameters field of the AlgorithmIdentifier within the SubjectPublicKeyInfo field of each certificate containing a KEA public key with an 80-bit parameter identifier (OCTET STRING), also known as the domain identifier. The domain identifier is computed in three steps:

(1) the KEA domain parameters (p, q, and g) are DER encoded using the Dss-Parms structure;

(2) a 160-bit SHA-1 hash is generated from the parameters; and

(3) the 160-bit hash is reduced to 80-bits by performing an "exclusive or" of the 80 high order bits with the 80 low order bits.

The resulting value is encoded such that the most significant byte of the 80-bit value is the first octet in the octet string. The Dss-Parms is provided above in Section 2.3.2.

A KEA public key, y, is conveyed in the subjectPublicKey BIT STRING such that the most significant bit (MSB) of y becomes the MSB of the BIT STRING value field and the least significant bit (LSB) of y becomes the LSB of the BIT STRING value field. This results in the following encoding:

```
BIT STRING tag;
BIT STRING length;
0 (indicating that there are zero unused bits in the final octet
of y); and
BIT STRING value field including y.
```

The key usage extension may optionally appear in a KEA certificate. If a KEA certificate includes the keyUsage extension, only the following values may be asserted:

```
keyAgreement;
encipherOnly; and
decipherOnly.
```

If present, the keyUsage extension MUST assert keyAgreement and MAY assert either encipherOnly and decipherOnly. The keyUsage extension MUST NOT assert both encipherOnly and decipherOnly.

2.3.5 ECDSA and ECDH Keys

This section identifies the preferred OID and parameter encoding for the inclusion of an ECDSA or ECDH public key in a certificate. The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in [X9.62]. ECDSA is the elliptic curve mathematical analog of the Digital Signature Algorithm [FIPS 186]. The Elliptic Curve Diffie Hellman (ECDH) algorithm is a key agreement algorithm defined in [X9.63].

ECDH is the elliptic curve mathematical analog of the Diffie-Hellman key agreement algorithm as specified in [X9.42]. The ECDSA and ECDH specifications use the same OIDs and parameter encodings. The ASN.1 object identifiers used to identify these public keys are defined in the following arc:

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

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

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

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

This OID is used in public key certificates for both ECDSA signature keys and ECDH encryption keys. The intended application for the key may be indicated in the key usage field (see [RFC 3280]). The use of a single key for both signature and encryption purposes is not recommended, but is not forbidden.

ECDSA and ECDH require use of certain parameters with the public key. The parameters may be inherited from the issuer, implicitly included through reference to a "named curve," or explicitly included in the certificate.

```
   EcpkParameters ::= CHOICE {
     ecParameters  ECParameters,
     namedCurve    OBJECT IDENTIFIER,
     implicitlyCA  NULL }
```

When the parameters are inherited, the parameters field SHALL contain implictlyCA, which is the ASN.1 value NULL. When parameters are specified by reference, the parameters field SHALL contain the named-Curve choice, which is an object identifier. When the parameters are explicitly included, they SHALL be encoded in the ASN.1 structure ECParameters:

```
ECParameters ::= SEQUENCE {
   version   ECPVer,          -- version is always 1
   fieldID   FieldID,         -- identifies the finite field over
                              -- which the curve is defined
   curve     Curve,           -- coefficients a and b of the
                              -- elliptic curve
   base      ECPoint,         -- specifies the base point P
                              -- on the elliptic curve
   order     INTEGER,         -- the order n of the base point
   cofactor  INTEGER OPTIONAL -- The integer h = #E(Fq)/n
   }

ECPVer ::= INTEGER {ecpVer1(1)}

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

FieldElement ::= OCTET STRING

ECPoint ::= OCTET STRING
```

The value of FieldElement SHALL be the octet string representation of a field element following the conversion routine in [X9.62], Section 4.3.3. The value of ECPoint SHALL be the octet string representation of an elliptic curve point following the conversion routine in [X9.62], Section 4.3.6. Note that this octet string may represent an elliptic curve point in compressed or uncompressed form.

Implementations that support elliptic curve according to this specification MUST support the uncompressed form and MAY support the compressed form.

The components of type ECParameters have the following meanings:

version specifies the version number of the elliptic curve parameters. It MUST have the value 1 (ecpVer1).

fieldID identifies the finite field over which the elliptic curve is defined. Finite fields are represented by values of the parameterized type FieldID, constrained to the values of the objects defined in the information object set FieldTypes. Additional detail regarding fieldID is provided below.

curve specifies the coefficients a and b of the elliptic curve E. Each coefficient is represented as a value of type FieldElement, an OCTET STRING. seed is an optional parameter used to derive the coefficients of a randomly generated elliptic curve.

base specifies the base point P on the elliptic curve. The base point is represented as a value of type ECPoint, an OCTET STRING.

order specifies the order n of the base point.

cofactor is the integer h = #E(Fq)/n. This parameter is specified as OPTIONAL. However, the cofactor MUST be included in ECDH public key parameters. The cofactor is not required to support ECDSA, except in parameter validation. The cofactor MAY be included to support parameter validation for ECDSA keys. Parameter validation is not required by this specification.

The AlgorithmIdentifier within SubjectPublicKeyInfo is the only place within a certificate where the parameters may be used. If the elliptic curve parameters are specified as implicitlyCA in the SubjectPublicKeyInfo AlgorithmIdentifier and the CA signed the subject certificate using ECDSA, then the certificate issuer's ECDSA parameters apply to the subject's ECDSA key. If the elliptic curve parameters are specified as implicitlyCA in the SubjectPublicKeyInfo AlgorithmIdentifier and the CA signed the certificate using a signature algorithm other than ECDSA, then clients MUST not make use of the elliptic curve public key.

```
FieldID ::= SEQUENCE {
   fieldType   OBJECT IDENTIFIER,
   parameters  ANY DEFINED BY fieldType }
```

FieldID is a SEQUENCE of two components, fieldType and parameters. The fieldType contains an object identifier value that uniquely identifies the type contained in the parameters.

The object identifier id-fieldType specifies an arc containing the object identifiers of each field type. It has the following value:

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

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

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

Prime-p ::= INTEGER    -- Field size p (p in bits)

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

Characteristic-two ::= SEQUENCE {
   m           INTEGER,                       -- Field size 2^m
   basis       OBJECT IDENTIFIER,
   parameters  ANY DEFINED BY basis }
```

The object identifier id-characteristic-two-basis specifies an arc containing the object identifiers for each type of basis for the characteristic-two finite fields. It has the following value:

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

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

```
gnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }

-- for gnBasis, the value of the parameters field is NULL

tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }

-- type of parameters field for tpBasis is Trinomial

Trinomial ::= INTEGER

ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }

-- type of parameters field for ppBasis is Pentanomial

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

The elliptic curve public key (an ECPoint which is an OCTET STRING) is mapped to a subjectPublicKey (a BIT STRING) as follows: the most significant bit of the OCTET STRING becomes the most significant bit of the BIT STRING, and the least significant bit of the OCTET STRING becomes the least significant bit of the BIT STRING. Note that this octet string may represent an elliptic curve point in compressed or uncompressed form. Implementations that support elliptic curve according to this specification MUST support the uncompressed form and MAY support the compressed form.

If the keyUsage extension is present in a CA or CRL issuer certificate which conveys an elliptic curve public key, any combination of the following values MAY be present:

digitalSignature;
nonRepudiation; and
keyAgreement.

If the keyAgreement value is present, either of the following values MAY be present:

encipherOnly; and
decipherOnly.

The keyUsage extension MUST NOT assert both encipherOnly and decipherOnly.

If the keyUsage extension is present in a CA certificate which conveys an elliptic curve public key, any combination of the following values MAY be present:

digitalSignature;
nonRepudiation;
keyAgreement;
keyCertSign; and
cRLSign.

As above, if the keyUsage extension asserts keyAgreement then it MAY assert either encipherOnly and decipherOnly. However, this specification RECOMMENDS that if keyCertSign or cRLSign is present, keyAgreement, encipherOnly, and decipherOnly SHOULD NOT be present.

3 ASN.1 Module

```
PKIX1Algorithms88 { iso(1) identified-organization(3) dod(6)
internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
id-mod-pkix1-algorithms(17) }

DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All;

-- IMPORTS NONE;

--
--   One-way Hash Functions
--

md2  OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549)
  digestAlgorithm(2) 2 }

md5  OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549)
  digestAlgorithm(2) 5 }

id-sha1  OBJECT IDENTIFIER ::= {
  iso(1) identified-organization(3) oiw(14) secsig(3)
  algorithms(2) 26 }

--
--   DSA Keys and Signatures
--

-- OID for DSA public key

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

-- encoding for DSA public key

DSAPublicKey ::= INTEGER  -- public key, y

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

```
   -- OID for DSA signature generated with SHA-1 hash

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

   -- encoding for DSA signature generated with SHA-1 hash

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

   --
   --   RSA Keys and Signatures
   --

   -- arc for RSA public key and RSA signature OIDs

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

   -- OID for RSA public keys

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

   -- OID for RSA signature generated with MD2 hash

   md2WithRSAEncryption OBJECT IDENTIFIER  ::=  { pkcs-1 2 }

   -- OID for RSA signature generated with MD5 hash

   md5WithRSAEncryption OBJECT IDENTIFIER  ::=  { pkcs-1 4 }

   -- OID for RSA signature generated with SHA-1 hash

   sha1WithRSAEncryption OBJECT IDENTIFIER  ::=  { pkcs-1 5 }

   -- encoding for RSA public key

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

```
   --
   --   Diffie-Hellman Keys
   --

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

   -- encoding for DSA public key

   DHPublicKey ::= INTEGER  -- public key, y = g^x mod p

   DomainParameters ::= SEQUENCE {
      p       INTEGER,            -- odd prime, p=jq +1
      g       INTEGER,            -- generator, g
      q       INTEGER,            -- factor of p-1
      j       INTEGER OPTIONAL,   -- subgroup factor, j>= 2
      validationParms  ValidationParms OPTIONAL }

   ValidationParms ::= SEQUENCE {
      seed             BIT STRING,
      pgenCounter      INTEGER }

   --
   --   KEA Keys
   --

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

   KEA-Parms-Id ::= OCTET STRING

   --
   --   Elliptic Curve Keys, Signatures, and Curves
   --

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

   FieldID ::= SEQUENCE {                    -- Finite field
      fieldType   OBJECT IDENTIFIER,
      parameters  ANY DEFINED BY fieldType }

   -- Arc for ECDSA signature OIDS

   id-ecSigType OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
```

```
   -- OID for ECDSA signatures with SHA-1

   ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType 1 }

   -- OID for an elliptic curve signature
   -- format for the value of an ECDSA signature value

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

   -- recognized field type OIDs are defined in the following arc

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

   -- where fieldType is prime-field, the parameters are of type Prime-p

   prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }

   Prime-p ::= INTEGER -- Finite field F(p), where p is an odd prime

   -- where fieldType is characteristic-two-field, the parameters are
   -- of type Characteristic-two

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

   Characteristic-two ::= SEQUENCE {
      m           INTEGER,                  -- Field size 2^m
      basis       OBJECT IDENTIFIER,
      parameters  ANY DEFINED BY basis }

   -- recognized basis type OIDs are defined in the following arc

   id-characteristic-two-basis OBJECT IDENTIFIER ::= {
        characteristic-two-field basisType(3) }

   -- gnbasis is identified by OID gnBasis and indicates
   -- parameters are NULL

   gnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }

   -- parameters for this basis are NULL

   -- trinomial basis is identified by OID tpBasis and indicates
   -- parameters of type Pentanomial

   tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
```

```
   -- Trinomial basis representation of F2^m
   -- Integer k for reduction polynomial xm + xk + 1

   Trinomial ::= INTEGER

   -- for pentanomial basis is identified by OID ppBasis and indicates
   -- parameters of type Pentanomial

   ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }

   -- Pentanomial basis representation of F2^m
   -- reduction polynomial integers k1, k2, k3
   -- f(x) = x**m + x**k3 + x**k2 + x**k1 + 1

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

   -- The object identifiers gnBasis, tpBasis and ppBasis name
   -- three kinds of basis for characteristic-two finite fields

   FieldElement ::= OCTET STRING             -- Finite field element

   ECPoint  ::= OCTET STRING                 -- Elliptic curve point

   -- Elliptic Curve parameters may be specified explicitly,
   -- specified implicitly through a "named curve", or
   -- inherited from the CA

   EcpkParameters ::= CHOICE {
      ecParameters  ECParameters,
      namedCurve    OBJECT IDENTIFIER,
      implicitlyCA  NULL }

   ECParameters  ::= SEQUENCE {         -- Elliptic curve parameters
      version   ECPVer,
      fieldID   FieldID,
      curve     Curve,
      base      ECPoint,                -- Base point G
      order     INTEGER,                -- Order n of the base point
      cofactor  INTEGER  OPTIONAL }     -- The integer h = #E(Fq)/n

   ECPVer ::= INTEGER {ecpVer1(1)}
```

```
   Curve  ::= SEQUENCE {
      a     FieldElement,            -- Elliptic curve coefficient a
      b     FieldElement,            -- Elliptic curve coefficient b
      seed  BIT STRING  OPTIONAL }

   id-publicKeyType OBJECT IDENTIFIER  ::= { ansi-X9-62 keyType(2) }

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

   -- Named Elliptic Curves in ANSI X9.62.

   ellipticCurve OBJECT IDENTIFIER ::= { ansi-X9-62 curves(3) }

   c-TwoCurve OBJECT IDENTIFIER ::= {
        ellipticCurve characteristicTwo(0) }

   c2pnb163v1  OBJECT IDENTIFIER  ::=  { c-TwoCurve  1 }
   c2pnb163v2  OBJECT IDENTIFIER  ::=  { c-TwoCurve  2 }
   c2pnb163v3  OBJECT IDENTIFIER  ::=  { c-TwoCurve  3 }
   c2pnb176w1  OBJECT IDENTIFIER  ::=  { c-TwoCurve  4 }
   c2tnb191v1  OBJECT IDENTIFIER  ::=  { c-TwoCurve  5 }
   c2tnb191v2  OBJECT IDENTIFIER  ::=  { c-TwoCurve  6 }
   c2tnb191v3  OBJECT IDENTIFIER  ::=  { c-TwoCurve  7 }
   c2onb191v4  OBJECT IDENTIFIER  ::=  { c-TwoCurve  8 }
   c2onb191v5  OBJECT IDENTIFIER  ::=  { c-TwoCurve  9 }
   c2pnb208w1  OBJECT IDENTIFIER  ::=  { c-TwoCurve 10 }
   c2tnb239v1  OBJECT IDENTIFIER  ::=  { c-TwoCurve 11 }
   c2tnb239v2  OBJECT IDENTIFIER  ::=  { c-TwoCurve 12 }
   c2tnb239v3  OBJECT IDENTIFIER  ::=  { c-TwoCurve 13 }
   c2onb239v4  OBJECT IDENTIFIER  ::=  { c-TwoCurve 14 }
   c2onb239v5  OBJECT IDENTIFIER  ::=  { c-TwoCurve 15 }
   c2pnb272w1  OBJECT IDENTIFIER  ::=  { c-TwoCurve 16 }
   c2pnb304w1  OBJECT IDENTIFIER  ::=  { c-TwoCurve 17 }
   c2tnb359v1  OBJECT IDENTIFIER  ::=  { c-TwoCurve 18 }
   c2pnb368w1  OBJECT IDENTIFIER  ::=  { c-TwoCurve 19 }
   c2tnb431r1  OBJECT IDENTIFIER  ::=  { c-TwoCurve 20 }

   primeCurve OBJECT IDENTIFIER ::= { ellipticCurve prime(1) }

   prime192v1  OBJECT IDENTIFIER  ::=  { primeCurve  1 }
   prime192v2  OBJECT IDENTIFIER  ::=  { primeCurve  2 }
   prime192v3  OBJECT IDENTIFIER  ::=  { primeCurve  3 }
   prime239v1  OBJECT IDENTIFIER  ::=  { primeCurve  4 }
   prime239v2  OBJECT IDENTIFIER  ::=  { primeCurve  5 }
   prime239v3  OBJECT IDENTIFIER  ::=  { primeCurve  6 }
   prime256v1  OBJECT IDENTIFIER  ::=  { primeCurve  7 }

   END
```

4 References

[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 186-2] Federal Information Processing Standards Publication (FIPS PUB) 186, Digital Signature Standard, 27 January 2000. [Supersedes FIPS PUB 186-1 dated 15 December 1998.]

[P1363] IEEE P1363, "Standard Specifications for Public-Key Cryptography", 2001.

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

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

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

[RFC 1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

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

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

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

[RFC 2313] Kaliski, B., "PKCS #1: RSA Encryption Version 1.5", RFC 2313, March 1998.

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

[RFC 3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC 3280] Housley, R., Polk, W., Ford, W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

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

[X.208] CCITT Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1), 1988.

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

[X9.42] ANSI X9.42-2000, "Public Key Cryptography for The Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography", December, 1999.

[X9.62] X9.62-1998, "Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", January 7, 1999.

[X9.63] ANSI X9.63-2001, "Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography", Work in Progress.

5 Security Considerations

This specification does not constrain the size of public keys or their parameters for use in the Internet PKI. However, the key size selected impacts the strength achieved when implementing cryptographic services. Selection of appropriate key sizes is critical to implementing appropriate security.

This specification does not identify particular elliptic curves for use in the Internet PKI. However, the particular curve selected impact the strength of the digital signatures. Some curves are cryptographically stronger than others!

In general, use of "well-known" curves, such as the "named curves" from ANSI X9.62, is a sound strategy. For additional information, refer to X9.62 Appendix H.1.3, "Key Length Considerations" and Appendix A.1, "Avoiding Cryptographically Weak Keys".

This specification supplements RFC 3280. The security considerations section of that document applies to this specification as well.

6 Intellectual Property Rights

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

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

7 Author Addresses:

Tim Polk
NIST
100 Bureau Drive, Stop 8930
Gaithersburg, MD 20899-8930
USA
EMail: tim.polk@nist.gov

Russell Housley
RSA Laboratories
918 Spring Knoll Drive
Herndon, VA 20170
USA
EMail: rhousley@rsasecurity.com

Larry Bassham
NIST
100 Bureau Drive, Stop 8930
Gaithersburg, MD 20899-8930
USA
EMail: lbassham@nist.gov

8. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.