PKIX Working Group Daniel R. L. Brown,
INTERNET−DRAFT Certicom Corp.
Expires July 6, 2006 January 6, 2006

Additional Algorithms and Identifiers
for use of Elliptic Curve Cryptography with PKIX
<draft−ietf−pkix−ecc−pkalgs−02.txt>

Status of this Memo

Abstract

This document gives additional algorithms and associated ASN.1 identifiers for elliptic curve cryptography (ECC) used with the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (PKIX). The algorithms and identifiers here are consistent with both ANSI X9.62−1998 and X9.63−2001, and shall be consistent with the forthcoming revisions of these documents. Consistency shall also be maintained with SEC1 and SEC2, and any revisions or successors of such documents.

Table of Contents

1 Introduction

This document supplements [RFC 3279], "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile "

This document specifies supplementary algorithm identifiers and ASN.1 [X.680] encoding formats for digital signatures and subject public keys used in the Internet X.509 Public Key Infrastructure (PKIX).

The supplementary formats specified are used to indicate the auxiliary functions, such as the new hash functions specified in [FIPS 180−2] including SHA−256, that are to be used with elliptic curve public keys.

Furthermore, this document specifies formats to indicate that an elliptic curve public key is to be restricted for use with an indicated set of elliptic curve cryptography algorithms.

Note: Previous standards [X9.62], [X9.63] and [SEC1] suggested that the extended key usage field could be used for purposes above. Because such a practice was regarded as improper, a new means to accomplish the objectives is being introduced both in this document and revisions of the standards above.

1.1 Terminology

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].

1.2 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is a family of cryptographic algorithms. Several algorithms, such as Diffie−Hellman (DH) key agreement and the Digital Signature Algorithm (DSA), have analogues in ECC. The analogy is that the cryptographic group is an elliptic curve group over a finite field rather the multiplicative group of (invertible) integers modulo a large prime.

Because an ECC groups and its elements are different from DH and DSA groups and elements, ECC requires a slightly different syntax from DSA and DH.

Because a single ECC public key in a certificated might potentially be used for multiple different ECC algorithms, a mechanism for indicating algorithm usage is important.

1.3 Algorithm Identifiers

The parameters field of the ASN.1 type AlgorithmIdentifier is optional when using ECC. When the parameters field is not used meaningfully, it SHOULD be absent, but MAY be NULL if it is necessary to interoperate with legacy implementations that do not support an optional parameters field. Absent and NULL parameters SHOULD both be accepted as valid and MUST then be considered to have the same meaning.

The following ASN.1 information object class helps to parameterize the AlgorithmIdentifier type with sets of legal values.

```
ALGORITHM ::= CLASS {
  &id     OBJECT IDENTIFIER UNIQUE,
  &Type   OPTIONAL
}
WITH SYNTAX { OID &id [PARMS &Type] }
```

The type AlgorithmIdentifier is parameterized to allow legal sets of values to be specified by constraining the type with an information object set.

```
AlgorithmIdentifier {ALGORITHM:IOSet} ::= SEQUENCE {
  algorithm   ALGORITHM.&id({IOSet}),
  parameters  ALGORITHM.&Type({IOSet}{@algorithm}) OPTIONAL
}
```

In practice, AlgorithmIdentifier is a sequence of an OID and an optional second field with syntax depending on the OID. In this document, the use of AlgorithmIdentifier will be constrained form. For example, when a hash function needs to be identified, a constrained form of AlgorithmIdentifier is used that only permits the OIDs for hash functions. The constraints also dictate the syntax for the parameters field for a given OID.

Note: Older syntax for AlgorithmIdentifier had a mandatory parameters field, which was customarily set to NULL when parameters field had nothing to convey. However, in the new syntax, in such situations the absent parameters are preferred to NULL parameters when the parameters field does not carry any meaning. This document specifies exactly what is permitted in the parameters field.

2 Auxiliary Functions

A number of different auxiliary functions are used in ECC. When two entities use an ECC algorithm in their communications with each other, they need to use matching auxiliary functions in order to successfully interoperate. Standards for ECC generally recommend or require certain choices of auxiliary functions, usually according to the elliptic curve key size in use. The following syntax helps to indicate, if needed, which auxiliary functions are to be used.

2.1 Hash Functions

Most notable among the auxiliary functions are hash functions, which are used in several different ways: message digesting for signatures, verifiably random domain parameter generation, building key derivation functions, building message authentication codes, as well as building random number generators.

The hash functions SHA−1, SHA−224, SHA−256, SHA−384 and SHA−512 can be used with ECC. They are specified in FIPS 180−2, Change Notice 1.

Hash functions are identified with the following object identifiers.

```
sha−1 OBJECT IDENTIFIER ::= { iso(1) identified−organization(3)
  oiw(14) secsig(3) algorithm(2) sha1(26) }

id−sha224 OBJECT IDENTIFIER ::= { joint−iso−itu−t(2) country(16)
  us(840) organization(1) gov(101) csor(3) nistalgorithm(4)
  hashalgs(2) 4 }

id−sha256 OBJECT IDENTIFIER ::= { joint−iso−itu−t(2) country(16)
  us(840) organization(1) gov(101) csor(3) nistalgorithm(4)
  hashalgs(2) 1 }

id−sha384 OBJECT IDENTIFIER ::= { joint−iso−itu−t(2) country(16)
  us(840) organization(1) gov(101) csor(3) nistalgorithm(4)
  hashalgs(2) 2 }

id−sha512 OBJECT IDENTIFIER ::= { joint−iso−itu−t(2) country(16)
  us(840) organization(1) gov(101) csor(3) nistalgorithm(4)
  hashalgs(2) 3 }
```

Others may be added.

The following information object set is used to constrain the AlgorithmIdentifier type for hash functions.

```
HashFunctions ALGORITHM ::= {
  {OID sha−1} | {OID sha−1 PARMS NULL } |
  {OID id−sha224} | {OID id−sha224 PARMS NULL } |
  {OID id−sha256} | {OID id−sha256 PARMS NULL } |
  {OID id−sha384} | {OID id−sha384 PARMS NULL } |
  {OID id−sha512} | {OID id−sha512 PARMS NULL } ,
  ... −− Additional hashes may be added
}
```

The constrained AlgorithmIdentifier syntax to identify a hash function is:

```
HashAlgorithm ::= AlgorithmIdentifier {{HashFunctions}}
```

The parameters SHOULD be absent but MAY be NULL.

2.2 Key Derivation Functions

<<< Rough version, only. Anticipate using a more flexible syntax in next update of this draft ... >>>

Crucial to key establishment, a Key Derivation Function (KDF) takes input of a raw elliptic curve point and other information such as identifiers, and then derives a key. A KDF helps to eliminate any structure from the key. (Elliptic curve points generally have some structure and cannot be regarded as pseudorandom.)

The KDF to use with ECC is specified in X9.63, except that the hash function SHA−1 can be replaced by one of SHA−1, SHA−224, SHA−256, SHA−384, or SHA−512. In particular, the KDF is determined entirely by the hash function it is built from, so the following syntax is adopted.

```
KeyDerivationFunction ::= HashAlgorithm
```

That certain protocols might use a different KDF, such as KDF1 in IEEE 1363−2000, only means that the specifications here are overridden in these protocols. Such KDFs ought to be deprecated. No ASN.1 syntax is given here to support such KDFs, making protocols that use such KDFs provide their own mechanisms to indicate use of them.

2.3 Key Wrap Functions

<<< To be added.>>>

Key wrap functions can be used to transform a key agreement scheme into a key transport scheme.

2.4 Message Authentication Codes

Some ECC algorithms use a Message Authentication Code (MAC), for example, as part of key confirmation.

<<< Surely these exist somewhere >>>

2.5 Key Confirmation Methods

<<< To be added. Unilateral, bilateral, etc.>>>

3 Elliptic Curve Domain Parameters

Elliptic curve domain parameters include the elliptic curve group used, as well as a particular element of this group, called base point or generator, and further includes the way the finite field elements in the elliptic curve points are represented. Elliptic domain parameters usually include further information such as order of the base point, a number called the cofactor, a value called seed which is used to select the curve, and possibly the base point, verifiably at random. Verifiably random domain parameters require an auxiliary hash function.

A few changes to elliptic curve domain parameters as originally specified in ANSI X9.62−2005 and ANSI X9.63−2001 mean that the corresponding ASN.1 syntax needs the following revisions.

The ASN.1 syntax to represent finite field elements and elliptic curve points remains unchanged.

The following new ASN.1 type provides the version numbers for explicitly specifying elliptic curve (EC) domain parameters.

```
SpecifiedECDomainVersion ::= INTEGER {
  ecdpVer1(1), ecdpVer2(2), ecdpVer3(3) }
```

The ASN.1 type for identifying an elliptic curve remains the same except the presence of its optional field is governed by the version number above.

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

The ASN.1 type for specifying EC domain parameters has been revised to include a field to identify the hash function used to generate the elliptic domain parameters verifiably at random, as follows.

```
SpecifiedECDomain ::= SEQUENCE {
  version  SpecifiedECDomainVersion (ecdpVer1 | ecdpVer2 | ecdpVer3),
  fieldID  FieldID {{FieldTypes}},
  curve    Curve,
  base     ECPoint,
  order    INTEGER,
  cofactor INTEGER OPTIONAL,
  hash     HashAlgorithm OPTIONAL −− New field
}
```

A version value of ecdpVer1 is used when either the domain parameters are not verifiably random or when the curve (not the base point) is verifiably random (from curve.seed). A version value of ecdpVer2 is used when the curve and the base point are both verifiably random (derived from curve.seed). A version value of ecdpVer3 is used when the base point, but not the curve, is verifiably random (derived from curve.seed).

If the hash is omitted then, the hash algorithm to be used is SHA−1.

The object identifiers for NIST recommended curves extend the object identifiers primeCurve and secgCurve whose values are

```
primeCurve OBJECT IDENTIFIER ::=
    { iso(1) member−body(2) us(840) 10045 curves(3) prime(1) }

secgCurve OBJECT IDENTIFIER ::=
    { iso(1) identified−organization(3) certicom(132) curve(0) }
```

The values of the object identifiers for the fifteen NIST recommended curves are

```
ansiX9p192r1 OBJECT IDENTIFIER ::= { primeCurve 1 }
ansiX9t163k1 OBJECT IDENTIFIER ::= { secgCurve  1 }
ansiX9t163r2 OBJECT IDENTIFIER ::= { secgCurve 15 }
ansiX9p224r1 OBJECT IDENTIFIER ::= { secgCurve 33 }
ansiX9t233k1 OBJECT IDENTIFIER ::= { secgCurve 26 }
ansiX9t233r1 OBJECT IDENTIFIER ::= { secgCurve 27 }
ansiX9p256r1 OBJECT IDENTIFIER ::= { primeCurve 7 }
ansiX9t283k1 OBJECT IDENTIFIER ::= { secgCurve 16 }
ansiX9t283r1 OBJECT IDENTIFIER ::= { secgCurve 17 }
ansiX9p384r1 OBJECT IDENTIFIER ::= { secgCurve 34 }
ansiX9t409k1 OBJECT IDENTIFIER ::= { secgCurve 36 }
ansiX9t409r1 OBJECT IDENTIFIER ::= { secgCurve 37 }
ansiX9p521r1 OBJECT IDENTIFIER ::= { secgCurve 35 }
ansiX9t571k1 OBJECT IDENTIFIER ::= { secgCurve 38 }
ansiX9t571r1 OBJECT IDENTIFIER ::= { secgCurve 39 }
```

The following information object class helps to constrain an field below to identify only a certain EC domain parameters.

```
ECDOMAIN ::= CLASS { &id OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX { ID &id }
```

The following information object set is used to constrain an AlgorithmIdentifier for identifying EC domain parameters.

```
ANSINamedECDomains ECDOMAIN ::= {
  { ID ansiX9p192r1 } | { ID ansiX9t163k1 } | { ID ansiX9t163r2 } |
  { ID ansiX9p224r1 } | { ID ansiX9t233k1 } | { ID ansiX9t233r1 } |
  { ID ansiX9p256r1 } | { ID ansiX9t283k1 } | { ID ansiX9t283r1 } |
  { ID ansiX9p384r1 } | { ID ansiX9t409k1 } | { ID ansiX9t409r1 } |
  { ID ansiX9p521r1 } | { ID ansiX9t571k1 } | { ID ansiX9t571r1 } ,
  ... -- Additional EC domain parameters may be added
}
```

The ASN.1 type for specifying elliptic curve domain parameters, whether explicitly, by name, or implicitly, is slightly revised as follows.

```
ECDomainParameters ::= CHOICE {
  specified   SpecifiedECDomain,
  named       ECDOMAIN.&id({ANSINamedECDomains}),
  implicitCA   NULL
}
```

4 ECC Algorithms

ECC algorithms can be identified using algorithm identifiers, in places such as PKIX certificates (and also in CMS).

In the new syntax here, the parameters field of these algorithm identifiers sometimes identifies the auxiliary functions.

4.1 Signature Schemes

4.1.1 ECDSA

To identify use of ECDSA with ASN.1, the auxiliary hash function for computing the message digest is necessary, which shall be implicit from the object identifier for ECDSA, and possibly as well as the corresponding public key, or shall be explicitly given in the parameters field, as detailed below.

The following object identifier serves as the root for further object identifier in this section.

```
id−ecSigType OBJECT IDENTIFIER ::=
{ iso(1) member−body(2) us(840) 10045 signatures(4) }
```

The following object identifier identifies SHA1 to be used for message digesting:

```
ecdsa−with−Sha1 OBJECT IDENTIFIER ::= { id−ecSigType sha1(1) }
```

The following new object identifier identifies the hash function to be used for message digesting is the one recommended for the public key size:

```
ecdsa−with−Recommended OBJECT IDENTIFIER ::=
{ id−ecSigType recommended(2) }
```

The recommended hash functions are given in the draft revision of X9.62, and is determined as follows. Among the hash functions SHA−1, SHA−224, SHA−256, SHA−384, SHA−512, the recommended one has the largest bit size that does not require bit truncation during the signing process. Bit truncation occurs the hash output bit length is greater than the bit length of n, the order of the base point G. (Note: even if bit truncation does not occur, modular reduction can occur.)

The following new object identifier identifies the hash function to be used for message digesting is the one specified in the parameters field of the algorithm identifier:

```
ecdsa-with-Specified OBJECT IDENTIFIER ::= {
  id-ecSigType specified(3)  }
```

The following new object identifiers directly identify the hash function to be used for message digesting.

```
ecdsa-with-Sha224 OBJECT IDENTIFIER ::=
{ id-ecSigType specified(3) 1 }
ecdsa-with-Sha256 OBJECT IDENTIFIER ::=
{ id-ecSigType specified(3) 2 }
ecdsa-with-Sha384 OBJECT IDENTIFIER ::=
{ id-ecSigType specified(3) 3 }
ecdsa-with-Sha512 OBJECT IDENTIFIER ::=
{ id-ecSigType specified(3) 4 }
```

The following information object set helps specify the legal set of algorithm identifiers for ECDSA.

```
ECDSAAlgorithmSet ALGORITHM ::= {
 {OID ecdsa-with-SHA1} |
 {OID ecdsa-with-SHA1 PARMS NULL} |
 {OID ecdsa-with-Recommended} |
 {OID ecdsa-with-Recommended PARMS NULL} |
 {OID ecdsa-with-Specified PARMS HashAlgorithm } |
 {OID ecdsa-with-Sha224} |
 {OID ecdsa-with-Sha256} |
 {OID ecdsa-with-Sha384} |
 {OID ecdsa-with-Sha512} ,
 ... -- More algorithms need to be added
}
```

The following type is the constrained AlgorithmIdentifier {} that identifies ECDSA:

```
ECDSAAlgorithm ::= AlgorithmIdentifier {{ECDSAAlgorithmSet}}
```

4.2 Key Agreement Schemes

The standard [X9.63] and draft standards [SP 800−56] and [DSEC1−v1.5] specify some ECC key agreement schemes. The standard [X9.63] also specifies some ASN.1 syntax, but this will be revised, as indicated below, in order to accommodate new hash functions such as SHA−256.

The following object identifiers are used as the root of other object identifiers that identify cryptographic schemes:

```
x9−63−scheme OBJECT IDENTIFIER ::= { iso(1) member−body(2)
us(840) ansi−x9−63(63) schemes(0) }

secg−scheme OBJECT IDENTIFIER ::= { iso(1)
identified−organization(3) certicom(132) schemes(1) }
```

4.2.1 1−Pass ECDH

<<< In progress ... >>>

In the 1−Pass Elliptic Curve Diffie−Hellman (ECDH) key agreement scheme, the initiator sends an ephemeral EC public key to the responder who has a static EC public key, typically in a certificate.

The following object identifiers from ANSI X9.63 identify the use of 1−Pass ECDH:

```
dhSinglePass−stdDH−sha1kdf OBJECT IDENTIFIER ::= {x9−63−scheme 2}
dhSinglePass−cofactorDH−sha1kdf OBJECT IDENTIFIER ::=
{x9−63−scheme 3}
dhSinglePass−cofactorDH−recommendedKDF OBJECT IDENTIFIER ::=
{secg−scheme 1}
dhSinglePass−cofactorDH−specifiedKDF OBJECT IDENTIFIER ::=
{secg−scheme 2}
```

The following information object set helps specify the legal set of algorithm identifiers for ECDH.

```
ECDHAlgorithmSet ALGORITHM ::= {
  {OID dhSinglePass−stdDH−sha1kdf} |
  {OID dhSinglePass−stdDH−sha1kdf PARMS NULL} |
  {OID dhSinglePass−cofactorDH−sha1kdf} |
  {OID dhSinglePass−cofactorDH−sha1kdf PARMS NULL} |
  {OID dhSinglePass−cofactorDH−recommendedKDF} |
  {OID dhSinglePass−cofactorDH−specifiedKDF
         PARMS KeyDerivationFunction} ,
   ... −− Future combinations may be added
}
```

The following type is the constrained AlgorithmIdentifier {} that legally identifies 1−Pass ECDH:

```
ECDHAlgorithm ::= AlgorithmIdentifier {{ECDHAlgorithmSet}}
```

4.2.2 Full and 1−Pass ECMQV

<<< In progress.>>>

In the Full and 1−Pass Elliptic Curve Menezes−Qu−Vanstone (ECMQV) key agreement schemes, both the initiator and responder have static EC public keys, typically in certificates, and the initiator sends an ephemeral EC public key to the responder. In Full ECMQV, the responder sends the initiator an ephemeral EC public key, but in 1−Pass ECMQV the sender does not.

The following object identifiers from ANSI X9.63 identify the

```
mqvSinglePass−sha1kdf OBJECT IDENTIFIER ::= {x9−63−scheme 16}
mqvSinglePass−recommendedKDF OBJECT IDENTIFIER ::= {secg−scheme 3}
mqvSinglePass−specifiedKDF OBJECT IDENTIFIER ::= {secg−scheme 4}
mqvFull−sha1kdf OBJECT IDENTIFIER ::= {x9−63−scheme 17}
mqvFull−recommendedKDF OBJECT IDENTIFIER ::= {secg−scheme 5}
mqvFull−specifiedKDF OBJECT IDENTIFIER ::= {secg−scheme 6}
```

The following information object set helps specify the legal set of algorithm identifiers for ECMQV.

```
ECMQVAlgorithmSet ALGORITHM ::= {
 {OID mqvSinglePass−sha1kdf} |
 {OID mqvSinglePass−recommendedKDF} |
 {OID mqvSinglePass−specifiedKDF PARMS KeyDerivationFunction} |
 {OID mqvFull−sha1kdf} |
 {OID mqvFull−recommendedKDF} |
 {OID mqvFull−specifiedKDF PARMS KeyDerivationFunction} ,
 ... −− Future combinations may be added
}
```

The following type is the constrained AlgorithmIdentifier {} that legally identifies 1−Pass and Full ECMQV:

```
ECMQVAlgorithm ::= AlgorithmIdentifier {{ECMQVAlgorithmSet}}
```

4.3 ECC Algorithm Set

The following information object set helps specify a legal set of ECC algorithms.

```
ECCAlgorithmSet ALGORITHM ::= {
  ECDSAAlgorithmSet |
  ECDHAlgorithmSet |
  ECMQVAlgorithmSet ,
  ... -- Future combinations may be added
}
```

The following type is the constrained AlgorithmIdentifier {} that legally identifies an ECC algorithm:

```
ECCAlgorithm ::= AlgorithmIdentifier {{ECCAlgorithmSet}}
```

The following type permits a sequence of ECC algorithm identifier to given.

```
ECCAlgorithms ::= SEQUENCE OF ECCAlgorithm
```

The order of the sequence SHOULD indicate an order of preference for which algorithm to used, where appropriate.

5 ECC Keys

Keys in ECC generally need to be associated with additional information such as domain parameters as well as, possibly, restrictions or preferences on algorithms that key can be used with.

5.1 Public Keys

Public keys are generally contained in certificates or stored in trusted memory, often in self−signed certificated format. Certificates are conveyed between parties or accessed from directories.

For certificates containing elliptic curve subject public keys, or certificates signed with elliptic curve issuer public keys using ECDSA, it is often necessary to identify the particular ECC algorithms and elliptic curve domain parameters that are used.

Certificates with ECC subject public keys can either restrict or not restrict the set of ECC algorithms with which they are used.

Unrestricted public keys are identified by the following OID:

```
id−ecPublicKey OBJECT IDENTIFIER ::= {
  iso(1) member−body(2) us(840) 10045 keyType(2) unrestricted(1)
}
```

This OID is used in an algorithm identifier as follows:

```
ecPublicKeyType ALGORITHM ::= {
  OID id−ecPublicKey PARMS ECDomainParameters
}
```

The following new syntax identifies ECC subject keys restricted to a certain subset of ECC algorithms. Firstly, the following OID is used:

```
id−ecPublicKeyRestricted OBJECT IDENTIFIER ::= {
  iso(1) member−body(2) us(840) 10045 keyType(2) restricted(2)
}
```

The following new syntax permits both elliptic curve domain parameters and a sequence of algorithm restrictions to be associated with an ECC public key:

```
ECPKRestrictions ::= SEQUENCE {
  ecDomain      ECDomainParameters,
  eccAlgorithms ECCAlgorithms
}
```

The new OID and new type are used in an algorithm identifier as follows:

```
ecPublicKeyTypeRestricted ALGORITHM ::= {
  OID id−ecPublicKeyRestricted PARMS ECPKRestrictions
}
```

The following information object set ECPKAlgorithmSet specifies the legal set of algorithm identifiers to identify an ECC public key:

```
ECPKAlgorithmSet ::= {
  ecPublicKeyType | ecPublicKeyTypeRestricted ,
  ... −− Further ECC public key types might be added
}
```

The following type uses the set above to constrain a algorithm identifier accordingly:

```
ECPKAlgorithm ::= AlgorithmIdentifier {ECPKAlgorithmSet}
```

In a PKIX certificate with an ECC subject public key, the SubjectPublicKeyInfo type shall use the following syntax:

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

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

5.2 Private Keys

<<< To be added. Perhaps unnecessary for PKIX. >>>

6 ASN.1 Module(s)

This module is extracted verbatim from a draft standard [DSEC1−v1.5], and is subject to revision.

```
DEFINITIONS EXPLICIT TAGS ::= BEGIN
  --
  -- EXPORTS ALL;
  --
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {    -- Finite field
  fieldType  FIELD-ID.&id({IOSet}),
  parameters FIELD-ID.&Type({IOSet}{@fieldType})
}
FIELD-ID ::= TYPE-IDENTIFIER
FieldTypes FIELD-ID ::= {
   { Prime-p            IDENTIFIED BY prime-field } |
   { Characteristic-two IDENTIFIED BY characteristic-two-field }
}
prime-field  OBJECT IDENTIFIER ::= { id-fieldType 1 }
Prime-p ::= INTEGER    -- Field of size p.
id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1)}
ansi-X9-62 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) 10045
}
characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }
Characteristic-two ::= SEQUENCE {
    m          INTEGER,    -- Field size 2^m
    basis      CHARACTERISTIC-TWO.&id({BasisTypes}),
    parameters CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}
CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
BasisTypes CHARACTERISTIC-TWO::= {
    { NULL        IDENTIFIED BY gnBasis } |
    { Trinomial   IDENTIFIED BY tpBasis } |
    { Pentanomial IDENTIFIED BY ppBasis },
    ...
}
gnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
id-characteristic-two-basis OBJECT IDENTIFIER ::= {
     characteristic-two-field basisType(3)
}
```

```
Trinomial ::= INTEGER
Pentanomial ::= SEQUENCE {
    k1 INTEGER,  -- k1 > 0
    k2 INTEGER,  -- k2 > k1
    k3 INTEGER   -- k3 > k2
}
FieldElement ::= OCTET STRING
ECDomainParameters{ECDOMAIN:IOSet} ::= CHOICE {
    specified  SpecifiedECDomain,
    named ECDOMAIN.&id({IOSet}),
    implicitCA   NULL
}
SpecifiedECDomain ::= SEQUENCE {
    version SpecifiedECDomainVersion(ecdpVer1 | ecdpVer2 |
  ecdpVer3, ...),
    fieldID  FieldID {{FieldTypes}},
    curve    Curve,
    base     ECPoint,
    order    INTEGER,
    cofactor INTEGER OPTIONAL,
    hash HashAlgorithm OPTIONAL,
    ...
}
SpecifiedECDomainVersion ::= INTEGER {
  ecdpVer1(1),
  ecdpVer2(2),
  ecdpVer3(3)
}
Curve ::= SEQUENCE {
    a    FieldElement,
    b    FieldElement,
    seed BIT STRING OPTIONAL
  -- Shall be present if used in SpecifiedECDomain with
  -- version equal to ecdpVer2 or ecdpVer3
}
ECPoint ::= OCTET STRING
ECDOMAIN ::= CLASS {
     &id  OBJECT IDENTIFIER UNIQUE
}
WITH SYNTAX { ID &id }
SECGCurveNames ECDOMAIN::= {
     ... -- named curves
}
```

```
HashAlgorithm ::= AlgorithmIdentifier {{ HashFunctions }}
HashFunctions ALGORITHM ::= {
  {OID sha−1} | {OID sha−1 PARMS NULL } |
  {OID id−sha224} | {OID id−sha224 PARMS NULL } |
  {OID id−sha256} | {OID id−sha256 PARMS NULL } |
  {OID id−sha384} | {OID id−sha384 PARMS NULL } |
  {OID id−sha512} | {OID id−sha512 PARMS NULL } ,
  ... −− Additional hash functions may be added in the future
}
sha−1 OBJECT IDENTIFIER ::= {iso(1) identified−organization(3)
oiw(14) secsig(3) algorithm(2) 26}
id−sha OBJECT IDENTIFIER ::= { joint−iso−itu−t(2) country(16) us(840)
organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) }
id−sha224 OBJECT IDENTIFIER ::= { id−sha 4 }
id−sha256 OBJECT IDENTIFIER ::= { id−sha 1 }
id−sha384 OBJECT IDENTIFIER ::= { id−sha 2 }
id−sha512 OBJECT IDENTIFIER ::= { id−sha 3 }
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm       AlgorithmIdentifier {{ECPKAlgorithms}},
    subjectPublicKey BIT STRING
}
AlgorithmIdentifier{ ALGORITHM:IOSet } ::= SEQUENCE {
    algorithm  ALGORITHM.&id({IOSet}),
    parameters ALGORITHM.&Type({IOSet}{@algorithm})
}
ALGORITHM ::= CLASS {
     &id   OBJECT IDENTIFIER UNIQUE,
     &Type OPTIONAL
}
     WITH SYNTAX { OID &id [PARMS &Type] }
ECPKAlgorithms ALGORITHM ::= {
  ecPublicKeyType |
  ecPublicKeyTypeRestricted |
  ecPublicKeyTypeSupplemented ,
    ...
}
ecPublicKeyType ALGORITHM ::= {
    OID id−ecPublicKey PARMS ECDomainParameters {{SECGCurveNames}}
}
id−ecPublicKey OBJECT IDENTIFIER ::= { id−publicKeyType 1 }
id−publicKeyType OBJECT IDENTIFIER ::= { ansi−X9−62 keyType(2) }
ecPublicKeyTypeRestricted ALGORITHM ::= {
  OID id−ecPublicKeyTypeRestricted PARMS ECPKRestrictions
}
id−ecPublicKeyTypeRestricted OBJECT IDENTIFIER ::= {
id−publicKeyType restricted(2) }
```

```
ECPKRestrictions ::= SEQUENCE {
  ecDomain ECDomainParameters {{ SECGCurveNames }},
  eccAlgorithms ECCAlgorithms
}
ECCAlgorithms ::= SEQUENCE OF ECCAlgorithm
ECCAlgorithm ::= AlgorithmIdentifier {{ECCAlgorithmSet}}
ecPublicKeyTypeSupplemented ALGORITHM ::= {
  OID id−ecPublicKeyTypeSupplemented PARMS ECPKSupplements
}
id−ecPublicKeyTypeSupplemented OBJECT IDENTIFIER ::= { iso(1)
identified−organization(3) certicom(132) schemes(1) supplementalPoints(0)}
ECPKSupplements ::= SEQUENCE {
  ecDomain ECDomainParameters {{ SECGCurveNames }},
  eccAlgorithms ECCAlgorithms,
  eccSupplements ECCSupplements
}
ECCSupplements ::= CHOICE {
  namedMultiples [0] NamedMultiples,
  specifiedMultiples [1] SpecifiedMultiples
}
NamedMultiples ::= SEQUENCE {
  multiples OBJECT IDENTIFIER,
  points SEQUENCE OF ECPoint
}
SpecifiedMultiples ::= SEQUENCE OF SEQUENCE {
  multiple INTEGER,
  point ECPoint
}
ECPrivateKey ::= SEQUENCE {
     version INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
     privateKey OCTET STRING,
     parameters [0] ECDomainParameters {{ SECGCurveNames }} OPTIONAL,
     publicKey  [1] BIT STRING OPTIONAL
}
ecdsa−with−SHA1 OBJECT IDENTIFIER ::=
  { id−ecSigType sha1(1)}
ecdsa−with−Recommended OBJECT IDENTIFIER ::=
{ id−ecSigType recommended(2) }
ecdsa−with−Specified OBJECT IDENTIFIER ::=
{ id−ecSigType specified(3)}
ecdsa−with−Sha224 OBJECT IDENTIFIER ::=
{ id−ecSigType specified(3) 1 }
ecdsa−with−Sha256 OBJECT IDENTIFIER ::=
{ id−ecSigType specified(3) 2 }
ecdsa−with−Sha384 OBJECT IDENTIFIER ::=
{ id−ecSigType specified(3) 3 }
ecdsa−with−Sha512 OBJECT IDENTIFIER ::=
{ id−ecSigType specified(3) 4 }
id−ecSigType OBJECT IDENTIFIER ::= { ansi−X9−62 signatures(4) }
```

```
ECDSAAlgorithmSet ALGORITHM ::= {
  {OID ecdsa-with-SHA1} |
  {OID ecdsa-with-SHA1 PARMS NULL} |
  {OID ecdsa-with-Recommended} |
  {OID ecdsa-with-Recommended PARMS NULL} |
  {OID ecdsa-with-Specified PARMS HashAlgorithm } |
  {OID ecdsa-with-Sha224} |
  {OID ecdsa-with-Sha256} |
  {OID ecdsa-with-Sha384} |
  {OID ecdsa-with-Sha512} ,
  ... -- More algorithms need to be added
}
ECCAlgorithmSet ALGORITHM ::= {
  ECDSAAlgorithmSet |
  ECDHAlgorithmSet |
  ECMQVAlgorithmSet |
  ECIESAlgorithmSet |
  ECWKTAlgorithmSet ,
  ...
}
ECDHAlgorithmSet ALGORITHM ::= {
  {OID dhSinglePass-stdDH-sha1kdf} |
  {OID dhSinglePass-stdDH-sha1kdf PARMS NULL} |
  {OID dhSinglePass-cofactorDH-sha1kdf} |
  {OID dhSinglePass-cofactorDH-sha1kdf PARMS NULL} |
  {OID dhSinglePass-cofactorDH-recommendedKDF} |
  {OID dhSinglePass-cofactorDH-specifiedKDF
   PARMS KeyDerivationFunction} ,
   ... -- Future combinations may be added
}
ECMQVAlgorithmSet ALGORITHM ::= {
  {OID mqvSinglePass-sha1kdf} |
  {OID mqvSinglePass-recommendedKDF} |
  {OID mqvSinglePass-specifiedKDF PARMS KeyDerivationFunction} |
  {OID mqvFull-sha1kdf} |
  {OID mqvFull-recommendedKDF} |
  {OID mqvFull-specifiedKDF PARMS KeyDerivationFunction} ,
  ... -- Future combinations may be added
}
x9-63-scheme OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) ansi-x9-63(63) schemes(0) }
secg-scheme OBJECT IDENTIFIER ::= { iso(1)
 identified-organization(3) certicom(132) schemes(1) }
dhSinglePass-stdDH-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 2}
dhSinglePass-cofactorDH-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 3}
mqvSinglePass-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 16}
mqvFull-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 17}
```

```
dhSinglePass−cofactorDH−recommendedKDF OBJECT IDENTIFIER ::=
 {secg−scheme 1}
dhSinglePass−cofactorDH−specifiedKDF OBJECT IDENTIFIER ::=
 {secg−scheme 2}
mqvSinglePass−recommendedKDF OBJECT IDENTIFIER ::= {secg−scheme 3}
mqvSinglePass−specifiedKDF OBJECT IDENTIFIER ::= {secg−scheme 4}
mqvFull−recommendedKDF OBJECT IDENTIFIER ::= {secg−scheme 5}
mqvFull−specifiedKDF OBJECT IDENTIFIER ::= {secg−scheme 6}
KeyDerivationFunction ::= HashAlgorithm
ECDSA−Sig−Value ::= SEQUENCE {
    r INTEGER,
    s INTEGER
}
END
```

7 References

[FIPS 180−2] U.S. Department of Commerce/National Institute of Standards and Technology. Secure Hash Standard (SHS), FIPS PUB 180−2, (http://csrc.nist.gov/fips/fips180−2.pdf)

[FIPS 186−2] U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS), FIPS PUB 186−2, January 2000. (http://csrc.nist.gov/fips/fips186−2.pdf)

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

[RFC 3278] S. Blake−Wilson, D. Brown and P. Lambert. Use of ECC Algorithms in CMS, April 2002.

[SEC1v1] Standards for Efficient Cryptography Group. SEC 1 − Elliptic Curve Cryptography, Version 1.0. September, 2000. (http://www.secg.org)

[DSEC1−v1.5] Standards for Efficient Cryptography Group. SEC 1 − Elliptic Curve Cryptography, Draft Version 1.5. February, 2005. (http://www.secg.org)

[SEC2] Standards for Efficient Cryptography Group. SEC 2 − Recommended Elliptic Curve Domain Parameters, Version 1.0. September, 2000. (http://www.secg.org)

[SP 800−56] NIST. Special Publication 800−56, Key Establishment Schemes, 2003.

[X9.62] American National Standard for Financial Services. ANSI X9.62−2005, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm. 1998.

[X9.63] American National Standard for Financial Services. ANSI X9.63−2001, Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography. November 2001.

8 Security Considerations

<<< To be added later. >>>

9 Acknowledgments

To be added later.

10 Authors' Addresses

Authors:

Daniel R. L. Brown
Certicom Corp.
dbrown@certicom.com

11 Full Copyright Statement