ECC in X.509

SECG X.509 WG Working Document

Working Group Draft, Version 0.2

Thursday, August 26, 1999

Contact: Peter de Rooij (Certicom Corp.), pderooij@certicom.com.

Table of Contents

1 Introduction

1.1 Goal

To enable and standardize the use of elliptic curve keys within the X.509 framework. The current scope is limited to those aspects of a Public Key Infrastructure (PKI) that are defined in X.509 [5]. That is, the certificate format and the format of certificate revocation lists (CRLs) are in scope. Management issues and supporting protocols (such as certificate request) are out of scope – these will be the subject of a follow-on document.

? *Is this the right scope?*

1.2 Objectives

- To specify how to indicate ECC keys and their usage within X.509 certificates.
- The chosen mechanism must be generic in the sense that wherever possible it should be applicable to any profile (specialization) of X.509 [5], such as PKIX [11] and X9.55 [10].
- Moreover, it should be compatible with any current initiatives to standardize ECC in X.509, in X.509 profiles, or outside of X.509.
- In addition, it should be flexible enough to be ready for more future profiles, for future extensions (new uses, new algorithms, etc.).
- Finally, the mechanism must be simple, and have the fewest options possible. (Of course, this nicely matches the other requirements.)

1.3 Current Situation

The X.509 standard [5] does not discuss algorithms; it only contains a placeholder for a list of supported algorithms and related parameters, with the intention that this be further specified by profiles based on the standard. Indeed, some profiles do, such as PKIX [11], X9.55 [10] and SET.

What is missing to achieve the above objectives is a general, universal way to specify ECC keys and all relevant aspects of the usage within X.509 [5].

On the other hand, several other standards and initiatives do define syntax for ECC keys and their usage. These include X9.62 [11], SEC1 [1], and the ECDSA extension proposal for PKIX [14].

1.4 What to specify

1.4.1 Areas of Impact

The choice of public key algorithm affects the following things in a certificate.

Certificate Signature Format – the certificate must uniquely identify the public key algorithm used by the CA for certificate signing. This involves the field that identifies this algorithm as well as the format of the signature itself.

Public Key Format – an X.509 certificate explicitly contains the certified subject public key. Therefore, the format of the public key must be specified. Again, this involves the field that identifies the algorithm (type) as well as the public key format itself. Note that by definition of its ASN.1 type, the algorithm type field may include the public key parameters (curve and base field).

Key Usage – certificates may list acceptable uses of the certified subject public key. This includes the mechanism (signature verification, encryption, key agreement, etc.), and potentially also the public key algorithms that may be used (ECDSA, ECDH, etc.). Only the latter is algorithm specific; the public key algorithm does not influence the encoding of the mechanism.

Each of these areas is discussed in a separate section in the remainder of this document. First, however, the ASN.1 types that must be described in these sections are listed (Section 1.4.2).

1.4.2 ASN.1 Types

A single ASN.1 type in an X.509 certificate identifies the choice of public key algorithm:

`AlgorithmIdentifier` – defines the (type of) public key algorithm used, possibly including parameters. It consists of an `algorithm` field and an optional `parameters` field. (Note: a given profile may mandate the presence of this optional field.)

The `algorithm` field may contain an algorithm type (such as RSA, discrete log, and elliptic curve) or an algorithm (such as RSA-PKCS1-with-SHA-1, DSA-with-SHA-1, and ECDSA-with-SHA-1). It is an OID (object identifier).

The `parameters` field (if given) ~~contains~~identifies the system parameters that define the "environment" in which public and private keys live. For ECC, the parameters object identifies the base field (incl. representation), the curve, the basis point, the generator, and some additional optional parameters (seed used for random generation of the curve, co-factor, etc.).

Note: If the `parameters` field is not given (i.e., is NULL), the system parameters are inherited from the CA certificate.

All other fields in a certificate that depend on the public key algorithm are binary (BIT STRING); the parsing is determined by an `AlgorithmIdentifier`:

- The `encrypted` field of the signed certificate (i.e., the certificate signature itself) is a BIT STRING, whose interpretation is determined by the `algorithmIdentifier` field, which is of type `AlgorithmIdentifier`. (PKIX calls this field `signatureValue`.)
- The `signature` field of the to-be-signed certificate duplicates the `algorithmIdentifier` field of the signed certificate. (PKIX calls this field `signatureAlgorithm`.)
- The `subjectPublicKey` field (inside the `subjectPublicKeyInfo` field of the to-be-signed certificate) is a BIT STRING, whose interpretation is determined by the `algorithm` field inside `subjectPublicKeyInfo`, which is of type `AlgorithmIdentifier`.
- The `supportedAlgorithms` attribute lists algorithms that may be supported with the given subject public key. The attribute contains an `AlgorithmIdentifier`, and optionally a `KeyUsage` and/or `CertificatePolicies` field.

(The public key algorithm does not (directly) affect the keyUsage and extendedKeyUsage extensions.)

The first two uses of AlgorithmIdentifier indicate a specific algorithm – the signature algorithm used to sign the certificate; the use in subjectPublicKey indicates an public key algorithm *type* – e.g., RSA, discrete log, or elliptic curve based. The use in supportedAlgorithms presumably would be used to indicate a specific algorithm (since the type already is indicated in the subjectPublicKey field).

1.5 Contributors

SECG ECC X.509 Certificate Working Group Members:

- Peter de Rooij (Certicom Corp.), Chair, pderooij@certicom.com.
- Michael Crerar (Diversinet Corp.), mcrerar@dvnet.com.
- Young Etheridge (Xcert International Inc.), yhe@xcert.com
- Don Johnson (Certicom Corp.), djohnson@certicom.com.
- Mark Shuttleworth (Thawte Consulting Inc.), marks@thawte.com
- William Whyte (Baltimore), william@baltimore.ie.

Simon Blake-Wilson, Bill Lattin and John Goyo (Certicom) provided additional comments and input.

1.6 To be done

! *Remove options: this version is based on X9.62 and PKIX; all irrelevant code and all options not supported by SEC1 must be removed!*

! *Replace references to PKIX or X9.62 by references to SEC1 or GEC1, where possible.*

! *Discussion of impact on CRLs, CRL signing, certificate request format (?).*

? *Open Issue: should this document be split into multiple documents? E.g., a separate document for each of: X.509, PKIX profile, X9.55 profile, …*

! *This document currently is a discussion document rather than a standard or guideline. This is to be changed in later editions.*

2 X.509 Syntax

2.1 Introduction

The X.509 standard itself does not go into details concerning formatting of signatures and public keys. Instead, it provides the means to *identify* the formatting of these fields. This is done by means of an `AlgorithmIdentifier`. As outlined in Section 1.4, the `AlgorithmIdentifier` allows for identification of both public keys and signatures.

This section details the X.509 syntax proper, and therefore focuses on the `AlgorithmIdentifier` only (identification of formatting).

The formatting itself (specification and interpretation of the bit strings that represent public keys and signatures) is specific to the algorithm. The specifications that currently exist for formatting of ECC based public keys and signatures are discussed in Section 3.

Section 4 links the *identification* and *formatting* into one proposed ECC X.509 syntax.

2.2 General Format of an X.509 Certificate

A Certificate contains a signature. The signature algorithm that was chosen for certificate signing determines its format. This is done by inserting an `AlgorithmIdentifier` in the signed message (the to-be-signed certificate) as well as in the signature itself.

The ASN.1 in X.509 for the certificate is:

```
CERTIFICATE   ::= SIGNED { SEQUENCE {
    version                 [0] Version DEFAULT v1,
    serialNuMBER                CertificateSerialNumer,
    signature                   AlgorithmIdentifier,
    issuer                      Name,
    validity                    Validity,
    subject                     Name,
    subjectPublickeyInfo        SubjectPublickeyInfo,
    issuerUniqueIdentifier  [1] IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier OPTIONAL,
    extensions                  Extensions OPTIONAL } }
```

The `signature` field specifies the public key algorithm used by the CA to sign this certificate.

The signature (i.e., the `SIGNED` macro) is defined as:

```
SIGNATURE {ToBeSigned}    ::= SEQUENCE {
    algorithmIdentifier   AlgorithmIdentifier,
    encrypted             ENCRYPTED-HASH {ToBeSigned}}

ENCRYPTED-HASH { ToBeSigned } ::= BIT STRING (CONSTRAINED BY {
    -- must be the result of applying a hashing procedure to the --
    -- DER encoded octets of a value of --
    ToBeSigned
    —-and then applying an encipherment procedure to these octets--}

SIGNED {ToBeSigned} ::= SEQUENCE {
    toBeSigned      ToBeSigned,
```

```
COMPONENTS OF   Signature { ToBeSigned }}
```

In other words, a signature is a sequence of the message to be signed, an AlgorithmIdentifier, and an encrypted hash (the signature itself). The AlgorithmIdentifier must accurately define the signature algorithm used.

2.3 X.509 Algorithm Identifier Syntax

The ASN.1 syntax for the type AlgorithmIdentifier in X.509 is:

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

ALGORITHM ::= TYPE-IDENTIFIER
```

In X.509, SupportedAlgorithms is not specified, and no list of supported ALGORITHMs (SupportedAlgorithms) is defined. This document does not define an exhaustive list for SupportedAlgorithms either. This is left to the applications based on this document. However, this document does define the *format* for AlgorithmIdentifier values for ECC keys.

2.4 PKIX Profile

The following syntax is the 1993 ASN.1 Syntax copied from Appendix B of PKIX [11]. First, PKIX rephrases the AlgorithmIdentifier syntax (renaming of ALGORITHM to ALGORITHM-ID).

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

The type ALGORITHM-ID is defined as follows:

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

This allows definition of SupportedAlgorithms:

```
SupportedAlgorithms  ALGORITHM-ID ::= {
    . . . , -- extensible
    rsaPublicKey | rsaSHA-1 | rsaMD5 | rsaMD2 |
    dssPublicKey | dsaSHA-1 | dhPublicKey }
```

This, finally, allows definition of AlgorithmIdentifiers, for example:

```
dssPublicKey ALGORITHM-ID ::= { OID id-dsa PARMS Dss-Params }
dsaSHA-1 ALGORITHM-ID ::= { OID id-dsa-with-sha1 }
. . .
id-dsa-with-sha1 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) x9-57(10040) x9algorithm 3 }
id-dsa OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) x9-57(10040) x9algorithm 1 }
. . .
Dss-Params ::= . . .  -- details omitted here
```

The former of these algorithm identifiers (dsspublicKey) is used in subjectPublicKeyInfo (indicating an algorithm *type*); the latter (dsaSHA-1) is used in the signature field (indicating a specific signature algorithm)

Note: the next release of the DSS (Digital Signature Standard, [4]) will specify a number of algorithms, including the DSA (Digital Signature Algorithm) as well as ECDSA and RSA. The prefix dss in the naming refers to DSA; this is a historical leftover from the first version of the DSS, that only contained DSA.

2.5 X9.55 Profile

This profile never goes into details concerning public key algorithms.

? *Correct?*

? *It does mention the Supported Algorithms Attribute as a "Directory Attribute". What does this mean? Is it signed (part of a certificate) or is it appended to a certificate only to facilitate identification of a particular certificate (in case a user has multiple certificates with different public keys, to be used with different algorithms)?*

? *Any other profiles I should mention?*

3 Existing ECC Syntax

3.1 ANSI X9.62 (ECDSA)

3.1.1 Introduction

This section compiles the ECC syntax from the ANSI X9.62 standard [11]. Those familiar with that standard may skip this section; it is supplied to render this document self-contained only.

3.1.2 Algorithm Identifier

The following syntax is copied from ANSI X9.62 [11]. In this standard, AlgorithmIdentifier is parameterized by {{ECPKAlgorithms}}:

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

The parameterized type AlgorithmIdentifier is then defined as:

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

(Note: the parameters are mandatory here; absence must be indicated by specifying NULL.)

The type ECPKAlgorithms is then defined as follows:

```
ECPKAlgorithms ALGORITHM ::= {
    ecPublicKeyType,
    . . . }

ecPublicKeyType ALGORITHM ::= { Parameters IDENTIFIED BY id-ecPublicKey }

ALGORITHM ::= TYPE_IDENTIFIER

id-ecPublicKey OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) 1 }
```

Thus, an ECC public key is identified as a sequence of the OID id-ecPublicKey and an instance of the type Parameters. The type Parameters is defined below, in 3.1.3.

An ECDSA signature is identified by an OID:

```
id-ecSigType      OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType 1 }
```

Note: the OIDs are formed as in PKIX [11]. In particular, ~~ecdsa-with-SHA1~~ecPublicKeyType is used to indicate a algorithm type in ~~SubjectPublciKeyInfo, id-ecSigType~~SubjectPublicKeyInfo, ecdsa-with-SHA1 is used to indicate a specific (signature) algorithm in the signature field.

However, the ecdsa-with-SHA1 OID is not declared as part of the supported algorithms ECPKAlgorithms. (In PKIX, both ~~dsspublicKey~~dssPublicKey (the DL analogue of ecPublicKeyType) and dsa-SHA1 (the ALGORITHM that contains the analogue of ecdsa-with-SHA1) are part of the supported algorithms SupportedAlgorithms.)

This is a consequence of the fact that ECDSA does not specify certificates, and therefore does not need to explicitly identify the ECDSA signature as an AlgorithmIdentifier. This specification will follow the PKIX model [11], as such an explicit identification is required in the current context.

? *Does this make sense?*

3.1.3 Parameters

3.1.3.1 Parameter Specification

There are three ways of indicating parameters:

- By explicit specification. This option is not recommended unless neither of the other (much more compact) options work.
- By reference. That is, by means of an OID for a specific parameter set. (This OID is constrained by the values in the set CurveNames, see below.)
- By inheritance. The parameters for a subject public key can be inherited from the issuer's certificate signing public key. (Note that these parameters in turn may be inherited from a higher level CA.) Inheritance is indicated by omission of the parameters in the ALGORITHM. That is, if no parameters are given, they are inherited from the parameters associated with the CA key.

This choice is encoded as:

```
Parameters ::= CHOICE {
    ecParameters  ECParameters,
    namedCurve    CURVES.&id({CurveNames}),
    implicitlyCA  NULL }
```

Below, the ECParameters type (Sections 3.1.3.2 and 3.1.3.3), the CURVES type, and the list of curves CurveNames (Section 3.1.3.4) are specified.

3.1.3.2 Domain Parameters

The ECParameters type explicitly specifies all elliptic curve domain parameters, as defined in ANSI X9.62 [11].

```
ECParameters ::= SEQUENCE {
    version   INTEGER { ecpVer1(1) (ecpVer1),
    fieldID   FieldID {{ FieldTypes }},
    curve     Curve,
    base      ECPoint,
    order     INTEGER,
    cofactor  INTEGER OPTIONAL,
    . . . }
```

As detailed in SEC1 [1], the inclusion of the cofactor is strongly recommended.

```
FieldElement ::= OCTET STRING
```

The value of FieldElement shall be the octet string representation of a field element following the conversion routine in X9.62, Section 4.3.1 [11].

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

ECPoint ::= OCTET STRING
```

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 [11].

The components of type ECParameters have the following meanings:

- version specifies the version number of the elliptic curve parameters. It shall have the value 1 for this version of the Standard. The notation above creates an INTEGER named ecpVer1 and gives it a value of one. It is used to constrain version to a single value.
- 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 (Section 3.1.3.3).
- curve specifies the coefficients *a* and *b* of the elliptic curve *E*. Each coefficient shall be 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 shall be represented as a value of type ECPoint, an OCTET STRING.
- order specifies the order *n* of the base point.
- cofactor is the integer $h = \#E(\mathbf{F})/n$. Note: This parameter is not used in ECDSA, except in parameter validation. It is included for compatibility with Elliptic Curve Key Agreement public key parameters. As detailed in SEC1 [1], the inclusion of the (optional) cofactor is strongly recommended.

The AlgorithmIdentifier within subjectPublicKeyInfo is the only place within a certificate where the parameters may be used. If the ECDSA algorithm parameters are absent from 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 ECDSA algorithm parameters are absent from the subjectPublicKeyInfo AlgorithmIdentifier and the CA signed the certificate using a signature algorithm other than ECDSA, then clients shall not validate the certificate.

3.1.3.3 Field Specification

```
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {
    fieldType     FIELD-ID.&id({IOSet}),
    parameters    FIELD-ID.&Type({IOSet}{@fieldType})  OPTIONAL }

FieldTypes FIELD-ID ::= {
    { Prime-p            IDENTIFIED BY prime-field } |
    { Characteristic-two IDENTIFIED BY characteristic-two-field },
    ...}

FIELD-ID ::= TYPE-IDENTIFIER
```

FieldID is a parameterized type composed of two components, fieldType and parameters. These components are specified by the fields &id and &Type, which form a template for defining sets of information objects, instances of the class FIELD-ID. This class is based on the useful information object class TYPE-IDENTIFIER, described in X.681 Annex A. In an instance of FieldID, "fieldType" will contain an object identifier value that uniquely identifies the type contained in "parameters". The effect of referencing "fieldType" in both components of the fieldID sequence is to tightly bind the object identifier and its type.

The information object set FieldTypes is used as the single parameter in a reference to type FieldID. FieldTypes contains two objects followed by the extension marker ("..."). Each object, which represents a finite field, contains a unique object identifier and its associated type. The values of these objects define all of the valid values that may appear in an instance of fieldID. The extension marker allows backward compatibility with future versions of this standard, which may define objects to represent additional kinds of finite fields.

The object identifier id-fieldType represents the root of a tree 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 }
characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }

Prime-p ::= INTEGER    -- Field size p (p in bits)
Characteristic-two ::= SEQUENCE {
    m               INTEGER,    -- Field size 2^m (m in bits)
    basis           CHARACTERISTIC-TWO.&id({BasisTypes}),
    parameters      CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis}) }

BasisTypes CHARACTERISTIC-TWO ::= {
   { NULL           IDENTIFIED BY onBasisgnBasis } |
   { Trinomial      IDENTIFIED BY tpBasis } |
   { Pentanomial    IDENTIFIED BY ppBasis },
   ... }

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

CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
```

The object identifier id-characteristic-two-basis represents the root of a tree 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 ~~onBasis,~~gnBasis, tpBasis and ppBasis name the three kinds of basis for characteristic-two finite fields defined by X9.62 [11]. They have the following values:

```
onBasisgnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
```

! *Bring in line with SEC1. Only tpBasis is recommended.*

3.1.3.4 Curves

CurveNames ~~lists~~must list all supported curves by their OIDs. ~~It is suggested to add the overarching list to GEC1 [2]. Each application then can support a subset of these curves, by appropriately redefining (trimming down) CurveNames.~~ X9.62 gives a specific list (not reproduced here).

```
CurveNames CURVES ::= {...}
. . .
        -- see GEC1 -- }

CURVES ::= CLASS {
    &id     OBJECT IDENTIFIER UNIQUE
```

```
} WITH SYNTAX { ID &id }
```

3.1.4 Public Keys

A public key is represented as the X.509 type SubjectPublicKeyInfo, using the following syntax:

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm          AlgorithmIdentifier,
    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, etc.; the least significant bit of the OCTET STRING becomes the least significant bit of the BIT STRING.

3.1.5 Signatures

When ECDSA and SHA-1 are used to sign an X.509 certificate or CRL, the signature shall be identified by the value ecdsa-with-SHA1, as defined below:

```
id-ecSigType OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType 1 }
```

When the ecdsa-with-SHA1 OID appears in the algorithm field of the ASN.1 type AlgorithmIdentifier, and the parameters field is a value of type NULL, the ECDSA parameters for signature verification must be obtained from other sources, such as the subjectPublicKeyInfo field of the certificate of the issuer.

When a digital signature is identified by the OID ecdsa-with-SHA1, the digital signature shall be ASN.1 encoded using the following syntax:

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

X.509 certificates and CRLs represent signatures as a bit string. Where a certificate or CRL is signed with ECDSA and SHA-1, the entire encoding of a value of ASN.1 type ECDSA-Sig-Value shall be the value of the bit string.

3.2 X9.63

! *to be added whenever X9.63 is ready…*

4 Proposed X.509 ECC Syntax

4.1 Introduction

4.1.1 Algorithm Identifiers in Public Key Info

We propose to use the ANSI X9.62 format [11] to indicate the EC public key *type*. That is, the X9.62 definition for `ecPublicKeyType`, when used in the `subjectPublicKeyInfo` `AlgorithmIdentifier`, indicates any EC public key, irrespective of the algorithm this key is used in.

Note: this `AlgorithmIdentifier` currently is the only one in a general standard. If and when more identifiers are added these may be added to indicate distinction between different types of EC algorithms. However, currently no such additional identifiers seem to be forthcoming (e.g., ANSI X9.63).

4.1.2 Algorithm Identifiers for Signature Type Identification

For the certificate signature, currently only ECDSA is supported; of course, the same formatting and the OID `ecdsa-with-SHA1` can be reused. Similarly, the enumeration is open to extension (addition of more signature algorithms). Extensions should always use existing OIDs to ease interoperability.

Note: If and when more algorithms are supported, the corresponding OIDs must be used. (None are currently planned to be added to SEC1.)

4.1.3 Summary

The `AlgorithmIdentifier` in `subjectPublicKeyInfo` is used to indicate the public key type. All EC public key types will use `ecPublicKeyType` from X9.62.

The certificate signature is identified by the `AlgorithmIdentifier` from the ECDSA signature standard (ANSI X9.62).

4.1.4 Section Outline

To describe both uses of `AlgorithmIdentifier`, the X.509 syntax [5] is adapted to encompass the identification of the ECDSA type as an `AlgorithmIdentifier`. This is detailed in Section 4.2. Furthermore, it is proposed to use the OIDs as specified in 4.3 (and in X9.62 [11]), and to specify the Parameters as in Section 4.4 (and in X9.62). Together, this defines the required `ALGORITHM`s.

4.2 Algorithm Identifiers

The syntax for `AlgorithmIdentifier` and `ALGORITHM` is copied from ~~X.509 [5]~~ ANSI X9.62 [11]; this is interoperable with PKIX [11] and ~~ANSI X9.62 [11]~~X.509 [5].

```
AlgorithmIdentifier ::= SEQUENCE {
    algorithm  ALGORITHM.&id({SupportedAlgorithms}),
    parameters ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL }
ALGORITHM ::= TYPE-IDENTIFIER
```

The algorithm identifiers are:

```
ecPublicKeyType ALGORITHM ::= { OID id-ecPublicKey PARMS Parameters }
ecdsa-SHA1 ALGORITHM ::= { OID ecdsa-with-SHA1 }
```

This allows definition of the SupportedAlgorithms:

```
SupportedAlgorithms  ALGORITHM ::= {
    . . . , -- extensible
    ecPublicKeyType | ecdsa-SHA1 }
```

! *Add ECAES, ECMQV, ECDH, etc. as soon as defined in X9.63. See Section 7.*

4.3 Algorithm Object Identifiers

Specify the relevant Algorithm OIDs for ECC support. These are already given in ANSI X9.62 (ECDSA) [11], as justified in Sections 5 and 6:

```
ansi-X9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10045 }
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { ansi-X9-62 sigType(4) 1 }
id-ecPublicKey OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) 1 }
```

The id-ecPublicKey OID is used in subjectPublicKeyInfo (indicating an algorithm *type*). The ecdsa-with-SHA1 OID is used in the signature field (indicating a specific algorithm).

! *Add the OIDs for ECAES, ECMQV, ECDH, etc. as soon as defined in X9.63. See Section 7.*

4.4 Parameters

Parameter definition follows and extends X9.62 (ECDSA), see 3.1.3. In particular, the inheritance is extended in the following sense. If parameters are inherited (indicated by omission of the parameters in the ALGORITHM), they are defined in the domainParametersTBD field of the CA if that exists, and otherwise they are inherited from the CA certificate signing public key (i.e., from the key used to verify the current certificate).

This is just an extension of the interpretation of the omission of the parameters in the ALGORITHM as far as the current certificate is concerned. The CA certificate may have a~~n~~ new domainParametersTBD field. In absence of that field, there is no difference with the definition of ECDSA. If that field is present in the CA certificate, there is a difference if the subject certificate omits the parameters in the ALGORITHM.

That is, parameters are defined:

- By explicit specification. This option is not recommended unless neither of the other (much more compact) options work.
- By reference. That is, by means of an OID for a specific parameter set.
- By inheritance. The parameters for a subject public key can be inherited from the issuer’s certificate.
    - If the issuer’s certificate contains a domainParametersTBD field, the parameters are inherited from that field.
    - If the issuer’s certificate does not contain a domainParametersTBD field, the parameters are inherited from the issuer’s certificate signing public key.

Inheritance is indicated by omission of the parameters in the ALGORITHM. That is, if no parameters are given, they are inherited from the parameters associated with the CA key as specified above.

`CurveNames` lists all supported curves by their OIDs. It is suggested to add the overarching list to GEC1 [2]. Each application then can support a subset of these curves, by appropriately redefining (trimming down) `CurveNames`.

```
CurveNames CURVES ::= {
    -- see GEC1 }

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

? *Incorporate the GEC1 list here?*

5 EC Certificate Signature Format

For ECC based certificates, currently only ECDSA (ANSI X9.62) signatures are supported. Therefore, the value for the `signature algorithmIdentifier` that identifies the certificate signing signature algorithm is taken from X9.62 [11], see 4.1.2:

```
ansi-X9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10045 }
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { ansi-X9-62 sigType(4) 1 }
ecdsa-SHA1 ALGORITHM ::= { OID ecdsa-with-SHA1 }
```

As in X9.62, the assumption is that the parameters are known to the verifier, as they are tied to (part of) the CA public key. That is, they have been provided by means external to the current certificate. (An example is: in the `subjectPublicKey` field (see Section 6) of a certificate of the CA that issued the current certificate.)

The signature format itself is defined in X9.62 as well [11]:

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

The signature in an X.509 certificate is a `BIT STRING`. As specified in X9.62 [11], the value of this `BIT STRING` shall be the entire encoding of a value of type `ECDSA-Sig-Value`.

? *Caveat: encoding rules (BER/PER/DER).*

Note: Future extension to other signature algorithms should follow the reasoning followed here: the signature standard defines both the OID and the signature format.

6 EC Public Key Format

The subject public key formatting (in a certificate) is specified by the `AlgorithmIdentifier` in the `subjectPublickeyInfo` field; see 4.1.1.

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

It is proposed to use the `AlgorithmIdentifier` value from ANSI X9.62 [11] for all ECC public keys, see 4.1.1. (Note: this may change in the future.)

Therefore, we propose to use the formatting of the public key as defined in [11] for all ECC public keys. The next section describes this format.

6.1 ECDSA Keys

The ECDSA standard (X9.62) contains an OID that is used as the algorithm identifier for ECC type algorithms [11]:

```
id-ecPublicKey OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) 1 }
```

This OID shall be used as detailed in 4.2:

```
ecPublicKeyType ALGORITHM ::= { OID id-ecPublicKey PARMS Parameters }
```

Section 3.1.3 specifies the encoding of `Parameters`.

Since the public key format depends on the public key *type* only and not on the specific signature algorithm (in this case: ECDSA), we can reuse the same OID for any elliptic curve public key.

The subject public key in an X.509 certificate is a `BIT STRING`. The value of this `BIT STRING` is defined as in X9.62 [11], as explained below.

The public key is a point on the elliptic curve defined by `Parameters`; that is, it is a value of type `ECPoint`. An `ECPoint` is an `OCTET STRING`. The `ECPoint` is encoded as an `OCTET STRING` as in Section 4.3.6 of X9.62 [11]; the `OCTET STRING` is encoded into a `BIT STRING` as defined in X9.62, Section 6.4. (That is, map the most significant bit to the most significant bit; map the least significant bit to the least significant bit.)

! *Change references to SEC1, and check if compliant with that…*

7 Supported Algorithm Indication

7.1 Introduction

The supportedAlgorithms ATTRIBUTE allows the specification of a list of algorithms, key usage and certificate policies that may be used with the subject public key. As such, there is no impact specific to ECC. However, use of this field for ECC keys requires the specification of OIDs for all algorithms supported by SEC1 [1].

Note that this section is not intended to recommend use of the supportedAlgorithms ATTRIBUTE; it is intended only to give guidance on its implementation when used. Note for example, that PKIX does not describe this attribute.

From X.509, Sec. 12.2.2.8:

```
supportedAlgorithms ATTRIBUTE ::= {
    WITH SYNTAX SupportedAlgorithm
    EQUALITY MATCHING RULE algorithmIdentifierMatch
    ID id-at-supportedAlgorithms }

SupportedAlgorithm ::= SEQUENCE {
    algorithmIdentifier               AlgorithmIdentifier,
    intendedUsage                 [0] KeyUsage OPTIONAL,
    intendedCertificatePolicies   [1] CertificatePoliciesSyntax OPTIONAL }
```

The supported AlgorithmIdentifiers are given in Section 7.2 below

? *Is this attribute inside the certificate or is it just in the directory, to ease searching? In the latter case, this chapter can go?*

7.2 OIDs

The following AlgorithmIdentifiers specify the algorithms currently supported by SEC1 [1]. SEC1 does not contain any (except ECDSA).

```
ansi-X9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10045 }
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { ansi-X9-62 sigType(4) 1 }
ecdsa-SHA1 ALGORITHM ::= { OID ecdsa-with-SHA1 }
```

! *To be added: ECAES, ECMQV, ECDH, … Level of detail to be determined.*

8 ASN.1 Module

! *Pseudo-ASN.1 -- to be cleaned up.*
An ASN.1 expert and some developers should have a close look at this!

```
ECCX509CertificateTBD {
    iso(1) identified-organization(3) certicom(132) module(1) ...}

DEFINITIONS EXPLICIT TAGS ::= BEGIN
EXPORTS All;
IMPORTS
    Certificate
        FROM AuthenticationFramework authenticationFramework
    Parameters, AlgorithmIdentifier, ecPublicKeyType, ecdsa-SHA1
        FROM ANSI-X9-62;
certicom-arc OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) certicom(132) }

SupportedAlgorithms  ALGORITHM ::= {
    ..., -- extensible
    ecPublicKeyType | ecdsa-SHA1 }

CurveNames CURVES ::= {
    ...,
    -- see GEC1 -- }

END
```

! *Same stuff without imports*

```
-- X.509 stuff
CERTIFICATE   ::= SIGNED { SEQUENCE {
    version                  [0] Version DEFAULT v1,
    serialNuMBER                 CertificateSerialNumer,
    signature                    AlgorithmIdentifier,
    issuer                       Name,
    validity                     Validity,
    subject                      Name,
    subjectPublickeyInfo         SubjectPublickeyInfo,
    issuerUniqueIdentifier   [1] IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueIdentifier  [2] IMPLICIT UniqueIdentifier OPTIONAL,
    extensions                   Extensions OPTIONAL } }
SIGNATURE {ToBeSigned}     ::= SEQUENCE {
    algorithmIdentifier   AlgorithmIdentifier,
    encrypted             ENCRYPTED-HASH {ToBeSigned}}

ENCRYPTED-HASH { ToBeSigned } ::= BIT STRING (CONSTRAINED BY {
    -- must be the result of applying a hashing procedure to the --
    -- DER encoded octets of a value of --
    ToBeSigned
    —-and then applying an encipherment procedure to these octets--}

SIGNED {ToBeSigned} ::= SEQUENCE {
    toBeSigned      ToBeSigned,
    COMPONENTS OF   Signature { ToBeSigned }}
```

```
-- ADDITION: support ECDSA.
ECDSA-Sig-Value ::= SEQUENCE {
    r   INTEGER,
    s   INTEGER }
-- END ADDITION

AlgorithmIdentifier ::= SEQUENCE {
    algorithm  ALGORITHM.&id({SupportedAlgorithms}),
    parameters ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL }
ALGORITHM ::= TYPE-IDENTIFIER

-- ADDITION: support Elliptic Curve PKs and ECDSA.
ecPublicKeyType ALGORITHM ::= { Parameters IDENTIFIED BY id-ecPublicKey }
id-ecPublicKey OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) 1 }

ecdsa-SHA1 ALGORITHM ::= { OID ecdsa-with-SHA1 }
id-ecSigType    OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType 1 }

SupportedAlgorithms  ALGORITHM ::= {
    . . . , -- extensible
    ecPublicKeyType | ecdsa-SHA1 }
-- END ADDITION

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

-- X9.62 stuff
Parameters ::= CHOICE {
    ecParameters  ECParameters,
    namedCurve    CURVES.&id({CurveNames}),
    implicitlyCA  NULL }

ECParameters ::= SEQUENCE {
    version   INTEGER { ecpVer1(1) (ecpVer1),
    fieldID   FieldID {{ FieldTypes }},
    curve     Curve,
    base      ECPoint,
    order     INTEGER,
    cofactor  INTEGER OPTIONAL,
    . . . }

FieldElement ::= OCTET STRING

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

ECPoint ::= OCTET STRING

FieldID { FIELD-ID:IOSet } ::= SEQUENCE {
    fieldType     FIELD-ID.&id({IOSet}),
    parameters    FIELD-ID.&Type({IOSet}{@fieldType})  OPTIONAL }

FieldTypes FIELD-ID ::= {
    { Prime-p           IDENTIFIED BY prime-field } |
```

```
    { Characteristic-two IDENTIFIED BY characteristic-two-field },
    ...}

FIELD-ID ::= TYPE-IDENTIFIER

id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1) }
prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }
characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }

Prime-p ::= INTEGER    -- Field size p (p in bits)
Characteristic-two ::= SEQUENCE {
    m               INTEGER,   -- Field size 2^m (m in bits)
    basis           CHARACTERISTIC-TWO.&id({BasisTypes}),
    parameters      CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis}) }

BasisTypes CHARACTERISTIC-TWO ::= {
   { NULL           IDENTIFIED BY onBasisgnBasis } |
   { Trinomial      IDENTIFIED BY tpBasis } |
   { Pentanomial    IDENTIFIED BY ppBasis },
   ... }

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

CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
id-characteristic-two-basis OBJECT IDENTIFIER ::= {
                   characteristic-two-field basisType(1) }
onBasisgnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
CurveNames CURVES ::= {
. . .
    -- see GEC1 -- }

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

References

[1] SEC1, *Elliptic Curve Cryptography*, Standards for Efficient Cryptography Group, Feb 16, 1999. Available from http://www.secg.org.

[2] GEC1, *Recommended Elliptic Curve Parameters*, Standards for Efficient Cryptography Group, Feb 16, 1999. Available from http://www.secg.org.

[3] FIPS 180-1, *Secure Hash Standard*, Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/N.I.S.T., National Information Service, Springfield, Virginia, April 17, 1995. Available from http://csrc.nist.gov/fips/fip180-1.pdf.

[4] FIPS 186-1, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-1, U.S. Department of Commerce/N.I.S.T., National Information Service, Springfield, Virginia, December 15, 1998. Available from http://csrc.nist.gov/fips/fips1861.pdf

[5] ITU-T Recommendation X.509, *Information Technology – Open Systems Interconnection – The Directory: Authentication Framework*. (Equivalent to ISO/IEC 9594-8.)

[6] ITU-T Recommendation X.680 (1994), *Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. (Equivalent to ISO/IEC 8824-1:1995.)

[7] ITU-T Recommendation X.681 (1994), *Information Technology – Abstract Syntax Notation One (ASN.1): Information Object Specification*. (Equivalent to ISO/IEC 8824-2:1995.)

[8] ITU-T Recommendation X.682 (1994), *Information Technology – Abstract Syntax Notation One (ASN.1): Constraint Specification*. (Equivalent to ISO/IEC 8824-3:1995.)

[9] ITU-T Recommendation X.683 (1994), *Information Technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications*. (Equivalent to ISO/IEC 8824-4:1995.)

[10]ANSI X9.55-1999, *Public Key Cryptography for the Financial Services Industry: Extensions to Public Key Certificates and Certificate Revocation Lists*, American Bankers Association, June 18, 1997.

[11]ANSI X9.62-1999, *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)*, American Bankers Association, 1999.

[12]ANSI X9.63-1999, *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American Bankers Association, Working Draft, April 20, 1999.

[13]R. Housley, W. Ford, et al., *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, Internet Engineering Task Force (IETF), Internet RFC 2459. This is part of the PKIX effort. The document is available from http://www.ietf.org/html.charters/pkix-charter.html.

[14]L. Bassham, D. Johnson and W. Polk, *Representation of Elliptic Curve Digital Signature Algorithm (ECDSA) Keys and Signatures in Internet X.509 Public Key Infrastructure Certificates*, Internet Draft, June 3, 1999. Available from http://www.ietf.org/html.charters/pkix-charter.html.