STANDARDS FOR EFFICIENT CRYPTOGRAPHY

SEC 1: Elliptic Curve Cryptography

Certicom Research
Contact: secg-talk@lists.certicom.com

September 20, 2000
Version 1.0

Contents

List of Tables

List of Figures

1 Introduction

1.1 Overview

This document specifies public-key cryptographic schemes based on elliptic curve cryptography (ECC). In particular, it specifies:

- signature schemes;
- encryption schemes; and
- key agreement schemes.

It also describes cryptographic primitives which are used to construct the schemes, and ASN.1 syntax for identifying the schemes.

The schemes are intended for general application within computer and communications systems.

1.2 Aim

The aim of this document is threefold.

Firstly to facilitate deployment of ECC by completely specifying efficient, well-established, and well-understood public-key cryptographic schemes based on ECC.

Secondly to encourage deployment of interoperable implementations of ECC by profiling existing standards like ANSI X9.62 [3], IEEE P1363 [40], and WAP WTLS [87], and draft standards like ANSI X9.63 [4] and IEEE P1363A [41], but restricting the options allowed in these standards to increase the likelihood of interoperability and to ensure conformance with all standards possible.

Thirdly to help ensure ongoing detailed analysis of ECC by cryptographers by clearly, completely, and publicly specifying baseline techniques.

1.3 Compliance

Implementations may claim compliance with the cryptographic schemes specified in this document provided the external interface (input and output) to the schemes is identical to the interface specified here. Internal computations may be performed as specified here, or may be performed via an equivalent sequence of operations.

Note that this compliance definition implies that conformant implementations must perform all the cryptographic checks included in the scheme specifications in this document. This is important because the checks are essential to the prevention of subtle attacks.

It is intended to make a validation system available so that implementors can check compliance with this document - see the SECG website, www.secg.org, for further information.

1.4 Document Evolution

This document will be reviewed every five years to ensure it remains up to date with cryptographic advances. The next scheduled review will therefore take place in September 2005.

Additional intermittent reviews may also be performed from time-to-time as deemed necessary by the Standards for Efficient Cryptography Group.

1.5 Intellectual Property

The reader's attention is called to the possibility that compliance with this document may require use of an invention covered by patent rights. By publication of this document, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. The patent holder(s) may have filed with the SECG a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. Additional details may be obtained from the patent holder and from the SECG website, www.secg.org.

1.6 Organization

This document is organized as follows.

The main body of the document focuses on the specification of public-key cryptographic schemes based on ECC. Section 2 describes the mathematical foundations fundamental to the operation of all the schemes. Section 3 provides the cryptographic components used to build the schemes. Sections 4, 5, and 6 respectively specify signature schemes, encryption schemes, and key agreement schemes based on ECC.

The appendices to the document provide additional relevant material. Appendix A gives a glossary of the acronyms and notation used as well as an explanation of the terms used. Appendix B elaborates some of the details of the main body — discussing implementation guidelines, making security remarks, and attributing references. Appendix C provides reference ASN.1 syntax for implementations to use to identify the schemes, and Appendix D lists the references cited in the document.

2 Mathematical Foundations

Use of each of the public-key cryptographic schemes described in this document involves arithmetic operations on an elliptic curve over a finite field. This section introduces the mathematical concepts necessary to understand and implement these arithmetic operations.

Section 2.1 discusses finite fields, Section 2.2 discusses elliptic curves over finite fields, and Section 2.3 describes the data types involved and the conventions used to convert between data types.

See Appendix B for a commentary on the contents on this section, including implementation discussion, security discussion, and references.

2.1 Finite Fields

Abstractly a finite field consists of a finite set of objects called field elements together with the description of two operations - addition and multiplication - that can be performed on pairs of field elements. These operations must possess certain properties.

It turns out that there is a finite field containing q field elements if and only if q is a power of a prime number, and furthermore that in fact for each such q there is precisely one finite field. The finite field containing q elements is denoted by $\mathbb{F}_q$.

Here only two types of finite fields $\mathbb{F}_q$ are used — finite fields $\mathbb{F}_p$ with $q = p$, p an odd prime which are called prime finite fields, and finite fields $\mathbb{F}_{2^m}$ with $q = 2^m$ for some $m \geq 1$ which are called characteristic 2 finite fields.

It is necessary to describe these fields concretely in order to precisely specify cryptographic schemes based on ECC. Section 2.1.1 describes prime finite fields and Section 2.1.2 describes characteristic 2 finite fields.

2.1.1 The Finite Field $\mathbb{F}_p$

The finite field $\mathbb{F}_p$ is the prime finite field containing p elements. Although there is only one prime finite field $\mathbb{F}_p$ for each odd prime p, there are many different ways to represent the elements of $\mathbb{F}_p$.

Here the elements of $\mathbb{F}_p$ should be represented by the set of integers:

$$\{0, 1, \ldots, p-1\}$$

with addition and multiplication defined as follows:

- Addition: If $a, b \in \mathbb{F}_p$, then $a + b = r$ in $\mathbb{F}_p$, where $r \in [0, p-1]$ is the remainder when the integer $a + b$ is divided by p. This is known as addition modulo p and written $a + b \equiv r \pmod{p}$.
- Multiplication: If $a, b \in \mathbb{F}_p$, then $a.b = s$ in $\mathbb{F}_p$, where $s \in [0, p-1]$ is the remainder when the integer ab is divided by p. This is known as multiplication modulo p and written $a.b \equiv s \pmod{p}$.

Addition and multiplication in $\mathbb{F}_p$ can be calculated efficiently using standard algorithms for ordinary integer arithmetic. In this representation of $\mathbb{F}_p$, the additive identity or zero element is the integer 0, and the multiplicative identity is the integer 1.

It is convenient to define subtraction and division of field elements just as it is convenient to define subtraction and division of integers. To do so, the additive inverse (or negative) and multiplicative inverse of a field element must be described:

- Additive inverse: If $a \in \mathbb{F}_p$, then the additive inverse $(-a)$ of a in $\mathbb{F}_p$ is the unique solution to the equation $a + x \equiv 0 \pmod{p}$.
- Multiplicative inverse: If $a \in \mathbb{F}_p$, $a \neq 0$, then the multiplicative inverse a^{-1} of a in $\mathbb{F}_p$ is the unique solution to the equation $a.x \equiv 1 \pmod{p}$.

Additive inverses and multiplicative inverses in $\mathbb{F}_p$ can be calculated efficiently. Multiplicative inverses are calculated using the extended Euclidean algorithm. Division and subtraction are defined in terms of additive and multiplicative inverses: $a - b \bmod p$ is $a + (-b) \bmod p$ and $a/b \bmod p$ is $a.(b^{-1}) \bmod p$.

Here the prime finite fields $\mathbb{F}_p$ used should have:

$$\lceil \log_2 p \rceil \in \{112, 128, 160, 192, 224, 256, 384, 521\}.$$

This restriction is designed to facilitate interoperability, while enabling implementers to deploy implementations which are efficient in terms of computation and communication since p is aligned with word size, and which are capable of furnishing all commonly required security levels. Inclusion of $\lceil \log_2 p \rceil = 521$ instead of $\lceil \log_2 p \rceil = 512$ is an anomaly chosen to align this document with other standards efforts - in particular with the U.S. government's recommended elliptic curve domain parameters [31]. See Appendix B for additional discussion on field selection.

2.1.2 The Finite Field $\mathbb{F}_{2^m}$

The finite field $\mathbb{F}_{2^m}$ is the characteristic 2 finite field containing 2^m elements. Although there is only one characteristic 2 finite field $\mathbb{F}_{2^m}$ for each power 2^m of 2 with $m \geq 1$, there are many different ways to represent the elements of $\mathbb{F}_{2^m}$.

Here the elements of $\mathbb{F}_{2^m}$ should be represented by the set of binary polynomials of degree $m-1$ or less:

$$\{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0 : a_i \in \{0,1\}\}$$

with addition and multiplication defined in terms of an irreducible binary polynomial $f(x)$ of degree m, known as the reduction polynomial, as follows:

- Addition: If $a = a_{m-1}x^{m-1} + \cdots + a_0$, $b = b_{m-1}x^{m-1} + \cdots + b_0 \in \mathbb{F}_{2^m}$, then $a + b = r$ in $\mathbb{F}_{2^m}$, where $r = r_{m-1}x^{m-1} + \cdots + r_0$ with $r_i \equiv a_i + b_i \pmod{2}$.

- Multiplication: If $a = a_{m-1}x^{m-1} + \cdots + a_0$, $b = b_{m-1}x^{m-1} + \cdots + b_0 \in \mathbb{F}_{2^m}$, then $a.b = s$ in $\mathbb{F}_{2^m}$, where $s = s_{m-1}x^{m-1} + \cdots + s_0$ is the remainder when the polynomial ab is divided by $f(x)$ with all coefficient arithmetic performed modulo 2.

Addition and multiplication in $\mathbb{F}_{2^m}$ can be calculated efficiently using standard algorithms for ordinary integer and polynomial arithmetic. In this representation of $\mathbb{F}_{2^m}$, the additive identity or zero element is the polynomial 0, and the multiplicative identity is the polynomial 1.

Again it is convenient to define subtraction and division of field elements. To do so the additive inverse (or negative) and multiplicative inverse of a field element must be described:

- Additive inverse: If $a \in \mathbb{F}_{2^m}$, then the additive inverse $(-a)$ of a in $\mathbb{F}_{2^m}$ is the unique solution to the equation $a + x = 0$ in $\mathbb{F}_{2^m}$.
- Multiplicative inverse: If $a \in \mathbb{F}_{2^m}$, $a \neq 0$, then the multiplicative inverse a^{-1} of a in $\mathbb{F}_{2^m}$ is the unique solution to the equation $a.x = 1$ in $\mathbb{F}_{2^m}$.

Additive inverses and multiplicative inverses in $\mathbb{F}_{2^m}$ can be calculated efficiently using the extended Euclidean algorithm. Division and subtraction are defined in terms of additive and multiplicative inverses: $a - b$ in $\mathbb{F}_{2^m}$ is $a + (-b)$ in $\mathbb{F}_{2^m}$ and a/b in $\mathbb{F}_{2^m}$ is $a.(b^{-1})$ in $\mathbb{F}_{2^m}$.

Here the characteristic 2 finite fields $\mathbb{F}_{2^m}$ used should have:

$$m \in \{113, 131, 163, 193, 233, 239, 283, 409, 571\}$$

and addition and multiplication in $\mathbb{F}_{2^m}$ should be performed using one of the irreducible binary polynomials of degree m in Table 1. As before this restriction is designed to facilitate interoperability while enabling implementers to deploy efficient implementations capable of meeting common security requirements.

Field	Reduction Polynomial(s)
$\mathbb{F}_{2^{113}}$	$f(x) = x^{113} + x^9 + 1$
$\mathbb{F}_{2^{131}}$	$f(x) = x^{131} + x^8 + x^3 + x^2 + 1$
$\mathbb{F}_{2^{163}}$	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
$\mathbb{F}_{2^{193}}$	$f(x) = x^{193} + x^{15} + 1$
$\mathbb{F}_{2^{233}}$	$f(x) = x^{233} + x^{74} + 1$
$\mathbb{F}_{2^{239}}$	$f(x) = x^{239} + x^{36} + 1$ or $x^{239} + x^{158} + 1$
$\mathbb{F}_{2^{283}}$	$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
$\mathbb{F}_{2^{409}}$	$f(x) = x^{409} + x^{87} + 1$
$\mathbb{F}_{2^{571}}$	$f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

Table 1: Representations of $\mathbb{F}_{2^m}$

The rule used to pick acceptable m's was: in each interval between integers in the set:

$$\{112, 128, 160, 192, 224, 256, 384, 512, 1024\},$$

if such an m exists, select the smallest prime m in the interval with the property that there exists a Koblitz curve whose order is 2 or 4 times a prime over $\mathbb{F}_{2^m}$; otherwise simply select the smallest prime m in the interval. (A Koblitz curve is an elliptic curve over $\mathbb{F}_{2^m}$ with $a, b \in \{0, 1\}$.) The inclusion of $m = 239$ is an anomaly chosen since it has already been widely used in practice. The inclusion of $m = 283$ instead of $m = 277$ is an anomaly chosen to align this document with other standards efforts - in particular with the U.S. government's recommended elliptic curve domain parameters [31]. Composite m was avoided to align this specification with other standards efforts and to address concerns about the security of elliptic curves defined over $\mathbb{F}_{2^m}$ with m composite. See Appendix B for additional discussion on field selection.

The rule used to pick acceptable reduction polynomials was: if a degree m binary irreducible trinomial:

$$f(x) = x^m + x^k + 1 \text{ with } m > k \geq 1$$

exists, use the irreducible trinomial with k as small as possible; otherwise use the degree m binary irreducible pentanomial:

$$f(x) = x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1 \text{ with } m > k_3 > k_2 > k_1 \geq 1$$

with (1) k_3 as small as possible, (2) k_2 as small as possible given k_3, and (3) k_1 as small as possible given k_3 and k_2. These polynomials enable efficient calculation of field operations. The second reduction polynomial at $m = 239$ is an anomaly chosen since it has been widely deployed.

2.2 Elliptic Curves

An elliptic curve over $\mathbb{F}_q$ is defined in terms of the solutions to an equation in $\mathbb{F}_q$. The form of the equation defining an elliptic curve over $\mathbb{F}_q$ differs depending on whether the field is a prime finite field or a characteristic 2 finite field.

Section 2.2.1 describes elliptic curves over prime finite fields, and Section 2.2.2 describes elliptic curves over characteristic 2 finite fields.

2.2.1 Elliptic Curves over $\mathbb{F}_p$

Let $\mathbb{F}_p$ be a prime finite field so that p is an odd prime number, and let $a, b \in \mathbb{F}_p$ satisfy $4.a^3 + 27.b^2 \not\equiv 0 \pmod{p}$. Then an elliptic curve $E(\mathbb{F}_p)$ over $\mathbb{F}_p$ defined by the parameters $a, b \in \mathbb{F}_p$ consists of the set of solutions or points $P = (x, y)$ for $x, y \in \mathbb{F}_p$ to the equation:

$$y^2 \equiv x^3 + a.x + b \pmod{p}$$

together with an extra point O called the point at infinity. The equation $y^2 \equiv x^3 + a.x + b \pmod{p}$ is called the defining equation of $E(\mathbb{F}_p)$. For a given point $P = (x_P, y_P)$, x_P is called the x-coordinate of P, and y_P is called the y-coordinate of P.

The number of points on $E(\mathbb{F}_p)$ is denoted by $\#E(\mathbb{F}_p)$. The Hasse Theorem states that:

$$p+1-2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p+1+2\sqrt{p}.$$

It is possible to define an addition rule to add points on E. The addition rule is specified as follows:

1. Rule to add the point at infinity to itself:

$$O+O=O.$$

2. Rule to add the point at infinity to any other point:

$$(x,y)+O=O+(x,y)=(x,y) \text{ for all } (x,y) \in E(\mathbb{F}_p).$$

3. Rule to add two points with the same x-coordinates when the points are either distinct or have y-coordinate 0:

$$(x,y)+(x,-y)=O \text{ for all } (x,y) \in E(\mathbb{F}_p)$$

— i.e. the negative of the point (x,y) is $-(x,y)=(x,-y)$.

4. Rule to add two points with different x-coordinates: Let $(x_1,y_1) \in E(\mathbb{F}_p)$ and $(x_2,y_2) \in E(\mathbb{F}_p)$ be two points such that $x_1 \neq x_2$. Then $(x_1,y_1)+(x_2,y_2)=(x_3,y_3)$, where:

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod{p},\ y_3 \equiv \lambda.(x_1-x_3)-y_1 \pmod{p}, \text{ and } \lambda \equiv \frac{y_2-y_1}{x_2-x_1} \pmod{p}.$$

5. Rule to add a point to itself (double a point): Let $(x_1,y_1) \in E(\mathbb{F}_p)$ be a point with $y_1 \neq 0$. Then $(x_1,y_1)+(x_1,y_1)=(x_3,y_3)$, where:

$$x_3 \equiv \lambda^2 - 2.x_1 \pmod{p},\ y_3 \equiv \lambda.(x_1-x_3)-y_1 \pmod{p}, \text{ and } \lambda \equiv \frac{3.x_1^2+a}{2.y_1} \pmod{p}.$$

The set of points on $E(\mathbb{F}_p)$ forms a group under this addition rule. Furthermore the group is abelian - meaning that $P_1+P_2=P_2+P_1$ for all points $P_1,P_2 \in E(\mathbb{F}_p)$. Notice that the addition rule can always be computed efficiently using simple field arithmetic.

Cryptographic schemes based on ECC rely on scalar multiplication of elliptic curve points. Given an integer k and a point $P \in E(\mathbb{F}_p)$, scalar multiplication is the process of adding P to itself k times. The result of this scalar multiplication is denoted $k \times P$ or kP. Scalar multiplication of elliptic curve points can be computed efficiently using the addition rule together with the double-and-add algorithm or one of its variants.

2.2.2 Elliptic Curves over $\mathbb{F}_{2^m}$

Let $\mathbb{F}_{2^m}$ be a characteristic 2 finite field, and let $a, b \in \mathbb{F}_{2^m}$ satisfy $b \neq 0$ in $\mathbb{F}_{2^m}$. Then a (non-supersingular) elliptic curve $E(\mathbb{F}_{2^m})$ over $\mathbb{F}_{2^m}$ defined by the parameters $a, b \in \mathbb{F}_{2^m}$ consists of the set of solutions or points $P = (x, y)$ for $x, y \in \mathbb{F}_{2^m}$ to the equation:

$$y^2 + x.y = x^3 + a.x^2 + b \text{ in } \mathbb{F}_{2^m}$$

together with an extra point O called the point at infinity. (Here the only elliptic curves over $\mathbb{F}_{2^m}$ of interest are non-supersingular elliptic curves.)

The number of points on $E(\mathbb{F}_{2^m})$ is denoted by $\#E(\mathbb{F}_{2^m})$. The Hasse Theorem states that:

$$2^m + 1 - 2\sqrt{2^m} \leq \#E(\mathbb{F}_{2^m}) \leq 2^m + 1 + 2\sqrt{2^m}.$$

It is again possible to define an addition rule to add points on E as it was in Section 2.2.1. The addition rule is specified as follows:

1. Rule to add the point at infinity to itself:
$$O + O = O.$$
2. Rule to add the point at infinity to any other point:
$$(x, y) + O = O + (x, y) = (x, y) \text{ for all } (x, y) \in E(\mathbb{F}_p).$$
3. Rule to add two points with the same x-coordinates when the points are either distinct or have x-coordinate 0:
$$(x, y) + (x, x + y) = O \text{ for all } (x, y) \in E(\mathbb{F}_p)$$
— i.e. the negative of the point (x, y) is $-(x, y) = (x, x + y)$.
4. Rule to add two points with different x-coordinates: Let $(x_1, y_1) \in E(\mathbb{F}_{2^m})$ and $(x_2, y_2) \in E(\mathbb{F}_{2^m})$ be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:
$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \text{ in } \mathbb{F}_{2^m},\ y_3 = \lambda.(x_1 + x_3) + x_3 + y_1 \text{ in } \mathbb{F}_{2^m}, \text{ and } \lambda \equiv \frac{y_1 + y_2}{x_1 + x_2} \text{ in } \mathbb{F}_{2^m}.$$
5. Rule to add a point to itself (double a point): Let $(x_1, y_1) \in E(\mathbb{F}_{2^m})$ be a point with $x_1 \neq 0$. Then $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, where:
$$x_3 = \lambda^2 + \lambda + a \text{ in } \mathbb{F}_{2^m},\ y_3 = x_1^2 + (\lambda + 1).x_3 \text{ in } \mathbb{F}_{2^m}, \text{ and } \lambda = x_1 + \frac{y_1}{x_1} \text{ in } \mathbb{F}_{2^m}.$$

The set of points on $E(\mathbb{F}_{2^m})$ forms an abelian group under this addition rule. Notice that the addition rule can always be computed efficiently using simple field arithmetic.

Cryptographic schemes based on ECC rely on scalar multiplication of elliptic curve points. As before given an integer k and a point $P \in E(\mathbb{F}_{2^m})$, scalar multiplication is the process of adding P to itself k times. The result of this scalar multiplication is denoted $k \times P$ or kP.

2.3 Data Types and Conversions

The schemes specified in this document involve operations using several different data types. This section lists the different data types and describes how to convert one data type to another.

Five data types are employed in this document: three types associated with elliptic curve arithmetic - integers, field elements, and elliptic curve points - as well as octet strings which are used to communicate and store information, and bit strings which are used by some of the primitives.

Throughout this document the above data types are regarded as abstract data types consisting of distinct sets of elements - so that, for example, an octet string is regarded as distinct from a bit string. This formalism helps to clarify the requirements placed on implementations and helps avoid subtle coding errors.

Frequently it is necessary to convert one of the data types into another - for example to represent an elliptic curve point as an octet string. The remainder of this section is devoted to describing how the necessary conversions should be performed.

Figure 1 illustrates which conversions are needed and where they are described.

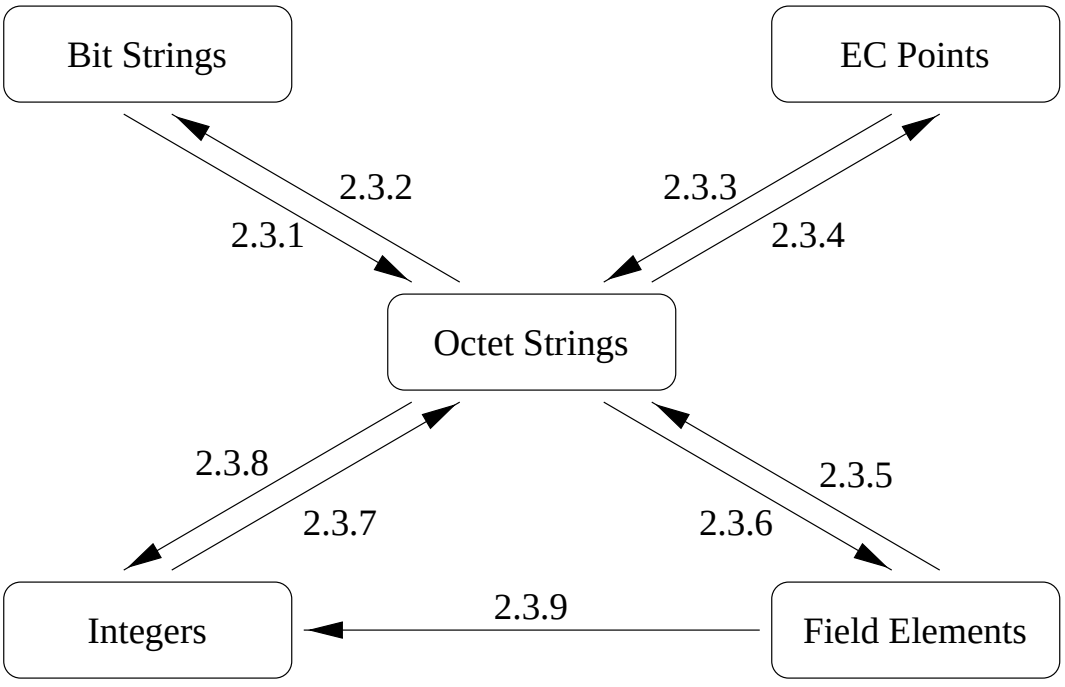

Figure 1: Converting between Data Types

2.3.1 BitString-to-OctetString Conversion

Bit strings should be converted to octet strings as described in this section. Informally the idea is to pad the bit string with 0's on the left to make its length a multiple of 8, then chop the result up into octets. Formally the conversion routine is specified as follows:

Input: A bit string B of length $blen$ bits.

Output: An octet string M of length $mlen = \lceil blen/8 \rceil$ octets.

Actions: Convert the bit string $B = B_0 B_1 \ldots B_{blen-1}$ to an octet string $M = M_0 M_1 \ldots M_{mlen-1}$ as follows:

1. For $0 < i \leq mlen - 1$, let:
$$M_i = B_{blen-8-8(mlen-1-i)} B_{blen-7-8(mlen-1-i)} \ldots B_{blen-1-8(mlen-1-i)}.$$
2. Let M_0 have its leftmost $8(mlen) - blen$ bits set to 0, and its rightmost $8 - (8(mlen) - blen)$ bits set to $B_0 B_1 \ldots B_{8-8(mlen)+blen-1}$.
3. Output M.

2.3.2 OctetString-to-BitString Conversion

Octet strings should be converted to bit strings as described in this section. Informally the idea is simply to view the octet string as a bit string instead. Formally the conversion routine is specified as follows:

Input: An octet string M of length $mlen$ octets.

Output: A bit string B of length $blen = 8(mlen)$ bits.

Actions: Convert the octet string $M = M_0 M_1 \ldots M_{mlen-1}$ to a bit string $B = B_0 B_1 \ldots B_{blen-1}$ as follows:

1. For $0 \leq i \leq mlen - 1$, set:
$$B_{8i} B_{8i+1} \ldots B_{8i+7} = M_i.$$
2. Output B.

2.3.3 EllipticCurvePoint-to-OctetString Conversion

Elliptic curve points should be converted to octet strings as described in this section. Informally, if point compression is being used, the idea is that the compressed y-coordinate is placed in the leftmost octet of the octet string along with an indication that point compression is on, and the x-coordinate is placed in the remainder of the octet string; otherwise if point compression is off, the leftmost octet indicates that point compression is off, and remainder of the octet string contains the x-coordinate followed by the y-coordinate. Formally the conversion routine is specified as follows:

Setup: Decide whether or not to represent points using point compression.

Input: A point P on an elliptic curve over $\mathbb{F}_q$ defined by the field elements a, b.

Output: An octet string M of length $mlen$ octets where $mlen = 1$ if $P = O$, $mlen = \lceil (\log_2 q)/8 \rceil + 1$ if $P \neq O$ and point compression is used, and $mlen = 2\lceil (\log_2 q)/8 \rceil + 1$ if $P \neq O$ and point compression is not used.

Actions: Convert P to an octet string $M = M_0 M_1 \ldots M_{mlen-1}$ as follows:

1. If $P = O$, output $M = 00_{16}$.
2. If $P = (x_P, y_P) \neq O$ and point compression is being used, proceed as follows:
   2.1. Convert the field element x_P to an octet string X of length $\lceil (\log_2 q)/8 \rceil$ octets using the conversion routine specified in Section 2.3.5.
   2.2. Derive from y_P a single bit $\tilde{y}_P$ as follows (this allows the y-coordinate to be represented compactly using a single bit):
      2.2.1. If $q = p$ is an odd prime, set $\tilde{y}_P = y_P \pmod 2$.
      2.2.2. If $q = 2^m$, set $\tilde{y}_P = 0$ if $x_P = 0$, otherwise compute $z = z_{m-1}x^{m-1} + \cdots + z_1 x + z_0$ such that $z = y_P . {x_P}^{-1}$ and set $\tilde{y}_P = z_0$.
   2.3. If $\tilde{y}_P = 0$, assign the value 02_{16} to the single octet Y. If $\tilde{y}_P = 1$, assign the value 03_{16} to the single octet Y.
   2.4. Output $M = Y \,||\, X$.
3. If $P = (x_P, y_P) \neq O$ and point compression is not being used, proceed as follows:
   3.1. Convert the field element x_P to an octet string X of length $\lceil (\log_2 q)/8 \rceil$ octets using the conversion routine specified in Section 2.3.5.
   3.2. Convert the field element y_P to an octet string Y of length $\lceil (\log_2 q)/8 \rceil$ octets using the conversion routine specified in Section 2.3.5.
   3.3. Output $M = 04_{16} \,||\, X \,||\, Y$.

2.3.4 OctetString-to-EllipticCurvePoint Conversion

Octet strings should be converted to elliptic curve points as described in this section. Informally the idea is that, if the octet string represents a compressed point, the compressed y-coordinate is recovered from the leftmost octet, the x-coordinate is recovered from the remainder of the octet string, and then the point compression process is reversed; otherwise the leftmost octet of the octet string is removed, the x-coordinate is recovered from the left half of the remaining octet string, and the y-coordinate is recovered from the right half of the remaining octet string. Formally the conversion routine is specified as follows:

Input: An elliptic curve over $\mathbb{F}_q$ defined by the field elements a, b, and an octet string M which is either the single octet 00_{16}, an octet string of length $mlen = \lceil (\log_2 q)/8 \rceil + 1$, or an octet string of length $mlen = 2\lceil (\log_2 q)/8 \rceil + 1$.

Output: An elliptic curve point P, or 'invalid'.

Actions: Convert M to an elliptic curve point P as follows:

1. If $M = 00_{16}$, output $P = O$.
2. If M has length $\lceil (\log_2 q)/8 \rceil + 1$ octets, proceed as follows:
   2.1. Parse $M = Y \| X$ as a single octet Y followed by $\lceil (\log_2 q)/8 \rceil$ octets X.
   2.2. Convert X to a field element x_P of $\mathbb{F}_q$ using the conversion routine specified in Section 2.3.6. Output 'invalid' and stop if the routine outputs 'invalid'.
   2.3. If $Y = 02$, set $\tilde{y}_P = 0$, and if $Y = 03$, set $\tilde{y}_P = 1$. Otherwise output 'invalid' and stop.
   2.4. Derive from x_P and $\tilde{y}_P$ an elliptic curve point $P = (x_P, y_P)$, where:
      2.4.1. If $q = p$ is an odd prime, compute the field element $\alpha \equiv {x_P}^3 + a.x_P + b \pmod{p}$, and compute a square root β of α modulo p. Output 'invalid' and stop if there are no square roots of α modulo p, otherwise set $y_P = \beta$ if $\beta \equiv \tilde{y}_P \pmod{2}$, and set $y_P = p - \beta$ if $\beta \not\equiv \tilde{y}_P \pmod{2}$.
      2.4.2. If $q = 2^m$ and $x_P = 0$, output $y_P = b^{2^{m-1}}$ in $\mathbb{F}_{2^m}$.
      2.4.3. If $q = 2^m$ and $x_P \neq 0$, compute the field element $\beta = x_P + a + b.{x_P}^{-2}$ in $\mathbb{F}_{2^m}$, and find an element $z = z_{m-1}x^{m-1} + \cdots + z_1 x + z_0$ such that $z^2 + z = \beta$ in $\mathbb{F}_{2^m}$. Output 'invalid' and stop if no such z exists, otherwise set $y_P = x_P.z$ in $\mathbb{F}_{2^m}$ if $z_0 = \tilde{y}_P$, and set $y_P = x_P.(z+1)$ in $\mathbb{F}_{2^m}$ if $z_0 \neq \tilde{y}_P$.
   2.5. Output $P = (x_P, y_P)$.
3. If M has length $2\lceil (\log_2 q)/8 \rceil + 1$ octets, proceed as follows:
   3.1. Parse $M = W \| X \| Y$ as a single octet W followed by $\lceil (\log_2 q)/8 \rceil$ octets X followed by $\lceil (\log_2 q)/8 \rceil$ octets Y.
   3.2. Check that $W = 04_{16}$. If $W \neq 04_{16}$, output 'invalid' and stop.
   3.3. Convert X to a field element x_P of $\mathbb{F}_q$ using the conversion routine specified in Section 2.3.6. Output 'invalid' and stop if the routine outputs 'invalid'.
   3.4. Convert Y to a field element y_P of $\mathbb{F}_q$ using the conversion routine specified in Section 2.3.6. Output 'invalid' and stop if the routine outputs 'invalid'.
   3.5. Check that $P = (x_P, y_P)$ satisfies the defining equation of the elliptic curve.
   3.6. Output $P = (x_P, y_P)$.

2.3.5 FieldElement-to-OctetString Conversion

Field elements should be converted to octet strings as described in this section. Informally the idea is that, if the field is $\mathbb{F}_p$, convert the integer to an octet string, and if the field is $\mathbb{F}_{2^m}$, view the coefficients

of the polynomial as a bit string with the highest degree term on the left and convert the bit string to an octet string. Formally the conversion routine is specified as follows:

Input: An element a of the field $\mathbb{F}_q$.

Output: An octet string M of length $mlen = \lceil \log_2 q/8 \rceil$ octets.

Actions: Convert a to an octet string $M = M_0 M_1 \ldots M_{mlen-1}$ as follows:

1. If $q = p$ is an odd prime, then a is an integer in the interval $[0, p-1]$. Convert a to M using the conversion routine specified in Section 2.3.7. Output M.

2. If $q = 2^m$, then $a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$ is a binary polynomial. Convert a to M as follows:

   2.1. For $0 < i \leq mlen - 1$, let:

   $$M_i = a_{7+8(mlen-1-i)} a_{6+8(mlen-1-i)} \cdots a_{8(mlen-1-i)}.$$

   2.2. Let M_0 have its leftmost $8(mlen) - m$ bits set to 0, and its rightmost $8 - (8(mlen) - m)$ bits set to $a_{m-1} a_{m-2} \ldots a_{8(mlen)-8}$.

   2.3. Output M.

2.3.6 OctetString-to-FieldElement Conversion

Octet strings should be converted to field elements as described in this section. Informally the idea is that, if the field is $\mathbb{F}_p$, convert the octet string to an integer, and if the field is $\mathbb{F}_{2^m}$, use the bits of the octet string as the coefficients of the binary polynomial with the rightmost bit as the constant term. Formally the conversion routine is specified as follows:

Input: An indication of the field $\mathbb{F}_q$ used and an octet string M of length $mlen = \lceil \log_2 q/8 \rceil$ octets.

Output: An element a in $\mathbb{F}_q$, or 'invalid'.

Actions: Convert $M = M_0 M_1 \ldots M_{mlen-1}$ with $M_i = M_i^0 M_i^1 \ldots M_i^7$ to a field element a as follows:

1. If $q = p$ is an odd prime, then a needs to be an integer in the interval $[0, p-1]$. Convert M to an integer a using the conversion routine specified in Section 2.3.8. Output 'invalid' and stop if a does not lie in the interval $[0, p-1]$, otherwise output a.

2. If $q = 2^m$, then a needs to be a binary polynomial of degree $m-1$ or less. Set the field element a to be $a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$ with:

   $$a_i = M_{mlen-1-\lfloor i/8 \rfloor}^{7-i+8(\lfloor i/8 \rfloor)}.$$

   Output 'invalid' and stop if the leftmost $8(mlen) - m$ bits of M_0 are not all 0, otherwise output a.

2.3.7 Integer-to-OctetString Conversion

Integers should be converted to octet strings as described in this section. Informally the idea is to represent the integer in binary then convert the resulting bit string to an octet string. Formally the conversion routine is specified as follows:

Input: A non-negative integer x together with the desired length $mlen$ of the octet string. It must be the case that:

$$2^{8(mlen)} > x.$$

Output: An octet string M of length $mlen$ octets.

Actions: Convert $x = x_{mlen-1}2^{8(mlen-1)} + x_{mlen-2}2^{8(mlen-2)} + \cdots + x_1 2^8 + x_0$ represented in base $2^8 = 256$ to an octet string $M = M_0 M_1 \ldots M_{mlen-1}$ as follows:

1. For $0 \leq i \leq mlen - 1$, set:
$$M_i = x_{mlen-1-i}.$$
2. Output M.

2.3.8 OctetString-to-Integer Conversion

Octet strings should be converted to integers as described in this section. Informally the idea is simply to view the octet string as the base 256 representation of the integer. Formally the conversion routine is specified as follows:

Input: An octet string M of length $mlen$ octets.

Output: An integer x.

Actions: Convert $M = M_0 M_1 \ldots M_{mlen-1}$ to an integer x as follows:

1. View M_i as an integer in the range $[0, 255]$ and set:
$$x = \sum_{i=0}^{mlen-1} 2^{8(mlen-1-i)} M_i.$$
2. Output x.

2.3.9 FieldElement-to-Integer Conversion

Field elements should be converted to integers as described in this section. Informally the idea is that, if the field is $\mathbb{F}_p$ no conversion is required, and if the field is $\mathbb{F}_{2^m}$ first convert the binary polynomial to an octet string then convert the octet string to an integer. Formally the conversion routine is specified as follows:

Input: An element a of the field $\mathbb{F}_q$.

Output: An integer x.

Actions: Convert the field element a to an integer x as follows:

1. If $q = p$ is an odd prime, then a must be an integer in the interval $[0, p-1]$. Output $x = a$.
2. If $q = 2^m$, then a must be a binary polynomial of degree $m-1$ — i.e. $a = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0$. Set:
$$x = \sum_{i=0}^{m-1} 2^i a_i.$$
Output x.

3 Cryptographic Components

This section describes the various cryptographic components that are used to build signature schemes, encryption schemes, and key agreement schemes later in this document.

See Appendix B for a commentary on the contents on this section, including implementation discussion, security discussion, and references.

3.1 Elliptic Curve Domain Parameters

The operation of each of the public-key cryptographic schemes described in this document involves arithmetic operations on an elliptic curve over a finite field determined by some elliptic curve domain parameters.

This section addresses the provision of elliptic curve domain parameters. It describes what elliptic curve domain parameters are, how they should be generated, and how they should be validated.

Two types of elliptic curve domain parameters may be used: elliptic curve domain parameters over $\mathbb{F}_p$, and elliptic curve domain parameters over $\mathbb{F}_{2^m}$. Section 3.1.1 describes elliptic curve domain parameters over $\mathbb{F}_p$, and Section 3.1.2 describes elliptic curve domain parameters over $\mathbb{F}_{2^m}$.

3.1.1 Elliptic Curve Domain Parameters over $\mathbb{F}_p$

Elliptic curve domain parameters over $\mathbb{F}_p$ are a sextuple:

$$T = (p, a, b, G, n, h)$$

consisting of an integer p specifying the finite field $\mathbb{F}_p$, two elements $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ defined by the equation:

$$E : y^2 \equiv x^3 + a.x + b \pmod{p},$$

a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_p)$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_p)/n$.

Elliptic curve domain parameters over $\mathbb{F}_p$ precisely specify an elliptic curve and base point. This is necessary to precisely define public-key cryptographic schemes based on ECC.

Section 3.1.1.1 describes how to generate elliptic curve domain parameters over $\mathbb{F}_p$, and Section 3.1.1.2 describes how to validate elliptic curve domain parameters over $\mathbb{F}_p$.

3.1.1.1 Elliptic Curve Domain Parameters over $\mathbb{F}_p$ Generation Primitive

Elliptic curve domain parameters over $\mathbb{F}_p$ should be generated as follows:

Input: The approximate security level in bits required from the elliptic curve domain parameters — this must be an integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$.

Output: Elliptic curve domain parameters over $\mathbb{F}_p$:

$$T = (p, a, b, G, n, h)$$

such that taking logarithms on the associated elliptic curve is believed to require approximately 2^t operations.

Actions: Generate elliptic curve domain parameters over $\mathbb{F}_p$ as follows:

1. Select a prime p such that $\lceil \log_2 p \rceil = 2t$ if $t \neq 256$ and such that $\lceil \log_2 p \rceil = 521$ if $t = 256$ to determine the finite field $\mathbb{F}_p$.

2. Select elements $a, b \in \mathbb{F}_p$ to determine the elliptic curve $E(\mathbb{F}_p)$ defined by the equation:

   $$E: \; y^2 \equiv x^3 + a.x + b \pmod{p},$$

   a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_p)$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_p)/n$, subject to the following constraints:

   - $4.a^3 + 27.b^2 \not\equiv 0 \pmod{p}$.
   - $\#E(\mathbb{F}_p) \neq p$.
   - $p^B \not\equiv 1 \pmod{n}$ for any $1 \leq B < 20$.
   - $h \leq 4$.

3. Output $T = (p, a, b, G, n, h)$.

This primitive allows any of the known curve selection methods to be used — for example the methods based on complex multiplication and the methods based on general point counting algorithms. However to foster interoperability it is strongly recommended that implementers use one of the elliptic curve domain parameters over $\mathbb{F}_p$ specified in SEC 2 [83]. See Appendix B for further discussion.

3.1.1.2 Validation of Elliptic Curve Domain Parameters over $\mathbb{F}_p$

Frequently it is either necessary or desirable for an entity using elliptic curve domain parameters over $\mathbb{F}_p$ to receive an assurance that the parameters are valid — that is that they satisfy the arithmetic requirements of elliptic curve domain parameters — either to prevent malicious insertion of insecure parameters, or to detect inadvertent coding or transmission errors.

There are four acceptable methods for an entity U to receive an assurance that elliptic curve domain parameters over $\mathbb{F}_p$ are valid. Only one of the methods must be supplied, although in many cases greater security may be obtained by carrying out more than one of the methods.

The four acceptable methods are:

1. U performs validation of the elliptic curve domain parameters over $\mathbb{F}_p$ itself using the validation primitive described in Section 3.1.1.2.1.

2. U generates the elliptic curve domain parameters over $\mathbb{F}_p$ itself using a trusted system using the primitive specified in Section 3.1.1.1.

3. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve domain parameters over $\mathbb{F}_p$ has performed validation of the parameters using the validation primitive described in Section 3.1.1.2.1.

4. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve domain parameters over $\mathbb{F}_p$ generated the parameters using a trusted system using the primitive specified in Section 3.1.1.1.

Usually when U accepts another party's assurance that elliptic curve domain parameters are valid, the other party is a CA.

3.1.1.2.1 Elliptic Curve Domain Parameters over $\mathbb{F}_p$ Validation Primitive

The elliptic curve domain parameters over $\mathbb{F}_p$ validation primitive should be used to check elliptic curve domain parameters over $\mathbb{F}_p$ are valid as follows:

Input: Elliptic curve domain parameters over $\mathbb{F}_p$:

$$T = (p, a, b, G, n, h),$$

along with an integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$ which is the approximate security level in bits required from the elliptic curve domain parameters.

Output: An indication of whether the elliptic curve domain parameters are valid or not — either 'valid' or 'invalid'.

Actions: Validate the elliptic curve domain parameters over $\mathbb{F}_p$ as follows:

1. Check that p is an odd prime such that $\lceil \log_2 p \rceil = 2t$ if $t \neq 256$ or such that $\lceil \log_2 p \rceil = 521$ if $t = 256$.

2. Check that a, b, x_G, and y_G are integers in the interval $[0, p-1]$.

3. Check that $4.a^3 + 27.b^2 \not\equiv 0 \pmod{p}$.

4. Check that ${y_G}^2 \equiv {x_G}^3 + a.x_G + b \pmod{p}$.

5. Check that n is prime.

6. Check that $h \leq 4$, and that $h = \lfloor (\sqrt{p}+1)^2/n \rfloor$.

7. Check that $nG = O$.

8. Check that $q^B \not\equiv 1 \pmod{n}$ for any $1 \leq B < 20$, and that $nh \neq p$.

9. If any of the checks fail, output 'invalid', otherwise output 'valid'.

Step 8 above excludes the known weak classes of curves which are susceptible to either the Menezes-Okamoto-Vanstone attack, or the Frey-Ruck attack, or the Semaev-Smart-Satoh-Araki attack. See Appendix B for further discussion.

If the elliptic curve domain parameters have been generated verifiably at random using SHA-1 as described in ANSI X9.62 [3], it may also be checked that a and b have been correctly derived from the random seed.

3.1.2 Elliptic Curve Domain Parameters over $\mathbb{F}_{2^m}$

Elliptic curve domain parameters over $\mathbb{F}_{2^m}$ are a septuple:

$$T = (m, f(x), a, b, G, n, h)$$

consisting of an integer m specifying the finite field $\mathbb{F}_{2^m}$, an irreducible binary polynomial $f(x)$ of degree m specifying the representation of $\mathbb{F}_{2^m}$, two elements $a, b \in \mathbb{F}_{2^m}$ specifying the elliptic curve $E(\mathbb{F}_{2^m})$ defined by the equation:

$$y^2 + x.y = x^3 + a.x^2 + b \text{ in } \mathbb{F}_{2^m},$$

a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_{2^m})$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_{2^m})/n$.

Elliptic curve domain parameters over $\mathbb{F}_{2^m}$ precisely specify an elliptic curve and base point. This is necessary to precisely define public-key cryptographic schemes based on ECC.

Section 3.1.2.1 describes how to generate elliptic curve domain parameters over $\mathbb{F}_{2^m}$, and Section 3.1.2.2 describes how to validate elliptic curve domain parameters over $\mathbb{F}_{2^m}$.

3.1.2.1 Elliptic Curve Domain Parameters over $\mathbb{F}_{2^m}$ Generation Primitive

Elliptic curve domain parameters over $\mathbb{F}_{2^m}$ should be generated as follows:

Input: The approximate security level in bits required from the elliptic curve domain parameters — this must be an integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$.

Output: Elliptic curve domain parameters over $\mathbb{F}_{2^m}$:

$$T = (m, f(x), a, b, G, n, h)$$

such that taking logarithms on the associated elliptic curve is believed to require approximately 2^t operations.

Actions: Generate elliptic curve domain parameters over $\mathbb{F}_{2^m}$ as follows:

1. Let t' denote the smallest integer greater than t in the set $\{64, 80, 96, 112, 128, 192, 256, 512\}$. Select $m \in \{113, 131, 163, 193, 233, 239, 283, 409, 571\}$ such that $2t < m < 2t'$ to determine the finite field $\mathbb{F}_{2^m}$.

2. Select a binary irreducible polynomial $f(x)$ of degree m from Table 1 in Section 2.1.2 to determine the representation of $\mathbb{F}_{2^m}$.

3. Select elements $a, b \in \mathbb{F}_{2^m}$ to determine the elliptic curve $E(\mathbb{F}_{2^m})$ defined by the equation:

$$E: y^2 + x.y = x^3 + a.x^2 + b \text{ in } \mathbb{F}_{2^m},$$

a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_{2^m})$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_{2^m})/n$, subject to the following constraints:

- $b \neq 0$ in $\mathbb{F}_{2^m}$.
- $\#E(\mathbb{F}_{2^m}) \neq 2^m$.
- $2^{mB} \not\equiv 1 \pmod{n}$ for any $1 \leq B < 20$.
- $h \leq 4$.

4. Output $T = (m, f(x), a, b, G, n, h)$.

This primitive also allows any of the known curve selection methods to be used. However to foster interoperability it is strongly recommended that implementers use one of the recommended elliptic curve domain parameters over $\mathbb{F}_{2^m}$ specified in SEC 2 [83]. See Appendix B for further discussion.

3.1.2.2 Validation of Elliptic Curve Domain Parameters over $\mathbb{F}_{2^m}$

Frequently it is either necessary or desirable for an entity using elliptic curve domain parameters over $\mathbb{F}_{2^m}$ to receive an assurance that the parameters are valid — that is that they satisfy the arithmetic requirements of elliptic curve domain parameters — either to prevent malicious insertion of insecure parameters, or to detect inadvertent coding or transmission errors.

There are four acceptable methods for an entity U to receive an assurance that elliptic curve domain parameters over $\mathbb{F}_{2^m}$ are valid. Only one of the methods must be supplied, although in many cases greater security may be obtained by carrying out more than one of the methods.

The four acceptable methods are:

1. U performs validation of the elliptic curve domain parameters over $\mathbb{F}_{2^m}$ itself using the validation primitive described in Section 3.1.2.2.1.

2. U generates the elliptic curve domain parameters over $\mathbb{F}_{2^m}$ itself using a trusted system using the primitive specified in Section 3.1.2.1.

3. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve domain parameters over $\mathbb{F}_{2^m}$ has performed validation of the parameters using the validation primitive described in Section 3.1.1.2.1.

4. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve domain parameters over $\mathbb{F}_{2^m}$ generated the parameters using a trusted system using the primitive specified in Section 3.1.2.1.

3.1.2.2.1 Elliptic Curve Domain Parameters over $\mathbb{F}_{2^m}$ Validation Primitive

The elliptic curve domain parameters over $\mathbb{F}_{2^m}$ validation primitive should be used to check elliptic curve domain parameters over $\mathbb{F}_{2^m}$ are valid as follows:

Input: Elliptic curve domain parameters over $\mathbb{F}_{2^m}$:

$$T = (m, f(x), a, b, G, n, h)$$

along with an integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$ which is the approximate security level in bits required from the elliptic curve domain parameters.

Output: An indication of whether the elliptic curve domain parameters are valid or not — either 'valid' or 'invalid'.

Actions: Validate the elliptic curve domain parameters over $\mathbb{F}_{2^m}$ as follows:

1. Let t' denote the smallest integer greater than t in the set $\{64, 80, 96, 112, 128, 192, 256, 512\}$. Check that m is an integer in the set $\{113, 131, 163, 193, 233, 239, 283, 409, 571\}$ such that $2t < m < 2t'$.

2. Check that $f(x)$ is a binary irreducible polynomial of degree m which is listed in Table 1 in Section 2.1.2.

3. Check that a, b, x_G, and y_G are binary polynomials of degree $m-1$ or less.

4. Check that $b \neq 0$ in $\mathbb{F}_{2^m}$.

5. Check that ${y_G}^2 + x_G.y_G = {x_G}^3 + a.{x_G}^2 + b$ in $\mathbb{F}_{2^m}$.

6. Check that n is prime.

7. Check that $h \leq 4$, and that $h = \lfloor (\sqrt{2^m}+1)^2/n \rfloor$.

8. Check that $nG = O$.

9. Check that $2^{mB} \not\equiv 1 \pmod{n}$ for any $1 \leq B < 20$, and that $nh \neq 2^m$.

10. If any of the checks fail, output 'invalid', otherwise output 'valid'.

Steps 1 and 9 above excludes the known weak classes of curves which are susceptible to either the Menezes-Okamoto-Vanstone attack, or the Frey-Ruck attack, or the Semaev-Smart-Satoh-Araki attack, or to attacks based on the Weil descent. See Appendix B for further discussion.

If the elliptic curve domain parameters have been generated verifiably at random using SHA-1 as described in ANSI X9.62 [3], it may also be checked that a and b have been correctly derived from the random seed.

3.2 Elliptic Curve Key Pairs

All the public-key cryptographic schemes described in this document use key pairs known as elliptic curve key pairs.

Given some elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$, an elliptic curve key pair (d, Q) associated with T consists of an elliptic curve secret key d which is an integer in the interval $[1, n-1]$, and an elliptic curve public key $Q = (x_Q, y_Q)$ which is the point $Q = dG$.

Section 3.2.1 describes how to generate elliptic curve key pairs, Section 3.2.2 describes how to validate elliptic curve public keys, and Section 3.2.3 describes how to partially validate elliptic curve public keys.

3.2.1 Elliptic Curve Key Pair Generation Primitive

Elliptic curve key pairs should be generated as follows:

Input: Valid elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$.

Output: An elliptic curve key pair (d, Q) associated with T.

Actions: Generate an elliptic curve key pair as follows:

1. Randomly or pseudorandomly select an integer d in the interval $[1, n-1]$.
2. Calculate $Q = dG$.
3. Output (d, Q).

3.2.2 Validation of Elliptic Curve Public Keys

Frequently it is either necessary or desirable for an entity using an elliptic curve public key to receive an assurance that the public key is valid — that is that it satisfies the arithmetic requirements of an elliptic curve public key — either to prevent malicious insertion of an invalid public key to enable attacks like small subgroup attacks, or to detect inadvertent coding or transmission errors.

There are four acceptable methods for an entity U to receive an assurance that an elliptic curve public key is valid. Only one of the methods must be supplied, although in many cases greater security may be obtained by carrying out more than one of the methods.

The four acceptable methods are:

1. U performs validation of the elliptic curve public key itself using the public key validation primitive described in Section 3.2.2.1.
2. U generates the elliptic curve public key itself using a trusted system.

3. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve public key has performed validation of the public key using the public key validation primitive described in Section 3.2.2.1.

4. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve public key generated the public key using a trusted system.

Usually when U accepts another party's assurance that an elliptic curve public key is valid, the other party is a CA who validated the public key during the certification process. Occasionally U may also receive assurance from another party other than a CA. For example, in the Station-to-Station protocol described in ANSI X9.63 [4], U receives an ephemeral public key from V. V is trusted with respect to U's use of the public key because U is attempting to establish a key with V and U only combines the public key with its own ephemeral key pair. It is therefore acceptable in this circumstance for U to accept assurance from V that the public key is valid because the public key is received in a signed message.

3.2.2.1 Elliptic Curve Public Key Validation Primitive

The elliptic curve public key validation primitive should be used to check an elliptic curve public key is valid as follows:

Input: Valid elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$, and an elliptic curve public key $Q = (x_Q, y_Q)$ associated with T.

Output: An indication of whether the elliptic curve public key is valid or not — either 'valid' or 'invalid'.

Actions: Validate the elliptic curve public key as follows:

1. Check that $Q \neq O$.

2. If T represents elliptic curve domain parameters over $\mathbb{F}_p$, check that x_Q and y_Q are integers in the range $[1, p-1]$, and that:
$$y_Q{}^2 \equiv x_Q{}^3 + a.x_Q + b \pmod{p}.$$

3. If T represents elliptic curve domain parameters over $\mathbb{F}_{2^m}$, check that x_Q and y_Q are binary polynomials of degree at most $m-1$, and that:
$$y_Q{}^2 + x_Q.y_Q = x_Q{}^3 + a.x_Q{}^2 + b \text{ in } \mathbb{F}_{2^m}.$$

4. Check that $nQ = O$.

5. If any of the checks fail, output 'invalid', otherwise output 'valid'.

In the above routine, steps 1, 2, and 3 check that Q is a point on E other than the point at infinity, and step 4 checks that Q is a scalar multiple of G.

3.2.3 Partial Validation of Elliptic Curve Public Keys

Sometimes it is sufficient for an entity using an elliptic curve public key to receive an assurance that the public key is partially valid, rather than 'fully' valid — here an elliptic curve public key Q is said to be partially valid if Q is a point on the associated elliptic curve but it is not necessarily the case that $Q = dG$ for some d.

The MQV key agreement scheme and the Diffie-Hellman scheme using the cofactor Diffie-Hellman primitive are both examples of schemes designed to provide security even when entities only check that the public keys involved are partially valid. (This feature is desirable because it means that the schemes enjoy a computational advantage in some circumstances over schemes like the Diffie-Hellman scheme with the 'standard' Diffie-Hellman primitive which require 'fully' valid public keys. The computational advantage stems from the fact that public key partial validation is more efficient than public key 'full' validation.)

There are four acceptable methods for an entity U to receive an assurance that an elliptic curve public key is partially valid. Only one of the methods must be supplied, although in many cases greater security may be obtained by carrying out more than one of the methods.

The four acceptable methods are:

1. U performs partial validation of the elliptic curve public key itself using the public key partial validation primitive described in Section 3.2.3.1.
2. U generates the elliptic curve public key itself using a trusted system.
3. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve public key has performed partial validation of the public key using the public key partial validation primitive described in Section 3.2.3.1.
4. U receives assurance in an authentic manner that a party trusted with respect to U's use of the elliptic curve public key generated the public key using a trusted system.

3.2.3.1 Elliptic Curve Public Key Partial Validation Primitive

The elliptic curve public key partial validation primitive should be used to check an elliptic curve public key is partially valid as follows:

Input: Valid elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$, and an elliptic curve public key $Q = (x_Q, y_Q)$ associated with T.

Output: An indication of whether the elliptic curve public key is partially valid or not — either 'valid' or 'invalid'.

Actions: Partially validate the elliptic curve public key as follows:

1. Check that $Q \neq O$.

2. If T represents elliptic curve domain parameters over $\mathbb{F}_p$, check that x_Q and y_Q are integers in the range $[1, p-1]$, and that:
$$y_Q{}^2 \equiv x_Q{}^3 + a.x_Q + b \pmod{p}.$$

3. If T represents elliptic curve domain parameters over $\mathbb{F}_{2^m}$, check that x_Q and y_Q are binary polynomials of degree at most $m-1$, and that:
$$y_Q{}^2 + x_Q.y_Q = x_Q{}^3 + a.x_Q{}^2 + b \text{ in } \mathbb{F}_{2^m}.$$

4. If any of the checks fail, output 'invalid', otherwise output 'valid'.

In the above routine, steps 1, 2, and 3 check that Q is a point on E other than the point at infinity.

3.3 Elliptic Curve Diffie-Hellman Primitives

This section specifies the elliptic curve Diffie-Hellman primitives which are the basis for the operation of the Elliptic Curve Integrated Encryption Scheme in Section 5.1, and the elliptic curve Diffie-Hellman scheme in Section 6.1.

Two primitives are specified: the elliptic curve Diffie-Hellman primitive and the elliptic curve cofactor Diffie-Hellman primitive. The basic idea of both primitives is the same — to generate a shared secret value from a private key owned by one entity U and a public key owned by another entity V so that if both entities execute the primitive with corresponding keys as input they will recover the same shared secret value.

However the two primitives are subtlely different: the elliptic curve Diffie-Hellman primitive is the straightforward analogue of the well-known Diffie-Hellman key agreement method, whereas the elliptic curve cofactor Diffie-Hellman primitive incorporates the cofactor into the calculation of the shared secret value to provide efficient resistance to attacks like small subgroup attacks.

The elliptic curve Diffie-Hellman primitive is specified in Section 3.3.1, and the elliptic curve cofactor Diffie-Hellman primitive is specified in Section 3.3.2.

3.3.1 Elliptic Curve Diffie-Hellman Primitive

U should employ the following process to calculate a shared secret value with V using the elliptic curve Diffie-Hellman primitive:

Input: The elliptic curve Diffie-Hellman primitive takes as input:

1. Valid elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$.

2. An elliptic curve private key d_U associated with T owned by U.

3. An elliptic curve public key Q_V associated with T purportedly owned by V.

The public key Q_V should be valid.

Output: A shared secret field element z, or 'invalid'.

Actions: Calculate a shared secret value as follows:

1. Compute the elliptic curve point $P = (x_P, y_P) = d_U Q_V$.
2. Check that $P \neq O$. If $P = O$, output 'invalid' and stop.
3. Output $z = x_P$ as the shared secret field element.

3.3.2 Elliptic Curve Cofactor Diffie-Hellman Primitive

U should employ the following process to calculate a shared secret value with V using the elliptic curve cofactor Diffie-Hellman primitive:

Input: The elliptic curve cofactor Diffie-Hellman primitive takes as input:

1. Valid elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$.
2. An elliptic curve private key d_U associated with T owned by U.
3. An elliptic curve public key Q_V associated with T purportedly owned by V.

The public key Q_V should at a minimum be partially valid.

Output: A shared secret field element z, or 'invalid'.

Actions: Calculate a shared secret value as follows:

1. Compute the elliptic curve point $P = (x_P, y_P) = h d_U Q_V$.
2. Check that $P \neq O$. If $P = O$, output 'invalid' and stop.
3. Output $z = x_P$ as the shared secret field element.

3.4 Elliptic Curve MQV Primitive

This section specifies the elliptic curve MQV primitive which is the basis for the operation of the elliptic curve MQV scheme specified in Section 6.2.

The basic idea of this primitive is to generate a shared secret value from two elliptic curve key pairs owned by one entity U and two elliptic curve public keys owned by another entity V so that if both entities execute the primitive with corresponding keys as input they will recover the same shared secret value.

U should employ the following process to calculate a shared secret value with V using the elliptic curve MQV primitive:

Input: The elliptic curve MQV primitive takes as input:

1. Valid elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$.
2. Two elliptic curve key pairs $(d_{1,U}, Q_{1,U})$ and $(d_{2,U}, Q_{2,U})$ associated with T owned by U.
3. Two elliptic curve public keys $Q_{1,V}$ and $Q_{2,V}$ associated with T purportedly owned by V.

The public keys $Q_{1,V}$ and $Q_{2,V}$ should at a minimum be partially valid.

Output: A shared secret field element z, or 'invalid'.

Actions: Calculate a shared secret value as follows:

1. Set $q = p$ if $T = (p, a, b, G, n, h)$, or $q = 2^m$ if $T = (m, f(x), a, b, G, n, h)$.
2. Compute an integer $\overline{Q_{2,U}}$ using $Q_{2,U} = (x_Q, y_Q)$ as follows:
   2.1. Convert x_Q to an integer x using the conversion routine specified in Section 2.3.9.
   2.2. Calculate:
   $$\overline{x} \equiv x \pmod{2^{\lceil (\log_2 n)/2 \rceil}}.$$
   2.3. Calculate:
   $$\overline{Q_{2,U}} = \overline{x} + 2^{\lceil (\log_2 n)/2 \rceil}.$$
3. Compute the integer:
   $$s \equiv d_{2,U} + \overline{Q_{2,U}} . d_{1,U} \pmod{n}.$$
4. Compute an integer $\overline{Q_{2,V}}$ using $Q_{2,V} = (x_Q{}', y_Q{}')$ as follows:
   4.1. Convert $x_Q{}'$ to an integer x' using the conversion routine specified in Section 2.3.9.
   4.2. Calculate:
   $$\overline{x'} \equiv x' \pmod{2^{\lceil (\log_2 n)/2 \rceil}}.$$
   4.3. Calculate:
   $$\overline{Q_{2,V}} = \overline{x'} + 2^{\lceil (\log_2 n)/2 \rceil}.$$
5. Compute the elliptic curve point:
   $$P = (x_P, y_P) = hs \times (Q_{2,V} + \overline{Q_{2,V}} Q_{1,V}).$$
6. Check that $P \neq O$. If $P = O$, output 'invalid' and stop.
7. Output $z = x_P$ as the shared secret field element.

3.5 Hash Functions

This section specifies the cryptographic hash functions supported in this document.

The hash functions are used by the key derivation functions specified in Section 3.6, and by the Elliptic Curve Digital Signature Algorithm specified in Section 4.1.

The hash functions will be used to calculate the hash value associated with an octet string.

The list of supported hash functions at this time is:

SHA-1

SHA-1 is specified in FIPS 180-1 [30]. It maps octet strings of length less than 2^{61} octets to hash values which are octet strings of length 20 octets. (Although only one hash function is supported at this time, it is planned that support for SHA-2 will be added as SHA-2 passes through the standardization procedures of ANSI and NIST so that a hash function which offers more than 80 bits of security is available to implementers.)

For clarity in the remainder of this section, the generic operation of the hash functions is described so that their use can be precisely specified later on.

Hash values should be calculated as follows:

Setup: Select one of the approved hash functions. Let $Hash$ denote the hash function chosen, *hashlen* denote the length in octets of hash values computed using $Hash$, and *hashmaxlen* denote the maximum length in octets of messages that can be hashed using $Hash$.

Input: The input to the hash function is an octet string M.

Output: The hash value H which is an octet string of length *hashlen* octets, or 'invalid'.

Actions: Calculate the hash value H as follows:

1. Check that M is less than *hashmaxlen* octets long — i.e. check that:

   $$||M|| < hashmaxlen.$$

   If $||M|| \geq hashmaxlen$, output 'invalid' and stop.

2. Convert M to a bit string $\overline{M}$ using the conversion routine specified in Section 2.3.2.

3. Calculate the hash value $\overline{H}$ corresponding to $\overline{M}$ using the selected hash function:

   $$\overline{H} = Hash(\overline{M}).$$

4. Convert $\overline{H}$ to an octet string H using the conversion routine specified in Section 2.3.1.

5. Output H.

3.6 Key Derivation Functions

This section specifies the key derivation functions supported in this document.

The key derivation functions are used by the Elliptic Curve Integrated Encryption Scheme specified in Section 5.1, and the key agreement schemes specified in Section 6.

The key derivation functions will be used to derive keying data from a shared secret octet string.

The list of supported key derivation functions at this time is:

ANSI-X9.63-KDF

ANSI-X9.63-KDF is the simple hash function construct described in ANSI X9.63 [4]. This key derivation function is described in Section 3.6.1 — partly for completeness since at the time of this edition ANSI X9.63 is only a draft standard, and partly so that the use of the key derivation function can be precisely described later on. Support for other key derivation functions may be added to future editions of this document.

3.6.1 ANSI X9.63 Key Derivation Function

Keying data should be calculated using ANSI-X9.63-KDF as follows:

Setup: Select one of the approved hash functions listed in Section 3.5. Let *Hash* denote the hash function chosen, *hashlen* denote the length in octets of hash values computed using *Hash*, and *hashmaxlen* denote the maximum length in octets of messages that can be hashed using *Hash*.

Input: The input to the key derivation function is:

1. An octet string Z which is the shared secret value.
2. An integer *keydatalen* which is the length in octets of the keying data to be generated.
3. (Optional) An octet string $SharedInfo$ which consists of some data shared by the entities intended to share the shared secret value Z.

Output: The keying data K which is an octet string of length *keydatalen* octets, or 'invalid'.

Actions: Calculate the keying data K as follows:

1. Check that $||Z|| + ||SharedInfo|| + 4 < hashmaxlen$. If $||Z|| + ||SharedInfo|| + 4 \geq hashmaxlen$, output 'invalid' and stop.
2. Check that $keydatalen < hashlen \times (2^{32} - 1)$. If $keydatalen \geq hashlen \times (2^{32} - 1)$, output 'invalid' and stop.
3. Initiate a 4 octet, big-endian octet string *Counter* as 00000001_{16}.

4. For $i = 1$ to $\lceil keydatalen/hashlen \rceil$, do the following:

   4.1. Compute:

   $$K_i = Hash(Z \| Counter \| [SharedInfo])$$

   using the selected hash function from the list of approved hash functions in Section 3.5.

   4.2. Increment *Counter*.

   4.3. Increment *i*.

5. Set K to be the leftmost *keydatalen* octets of:

   $$K_1 \| K_2 \| \ldots \| K_{\lceil keydatalen/hashlen \rceil}.$$

6. Output K.

3.7 MAC schemes

This section specifies the MAC schemes supported in this document.

The MAC schemes will be used by the Elliptic Curve Integrated Encryption Scheme specified in Section 5.1.

MAC schemes are designed to be used by two entities — a sender U and a recipient V — when U wants to send a message M to V in an authentic manner and V wants to verify the authenticity of M.

Here the MAC schemes are described in terms of a tagging operation, a tag checking operation, and associated setup and key deployment procedures. U and V should use the schemes as follows when they want to communicate. First U and V should use the setup and key deployment procedures to establish which options to use the scheme with, and to create a shared secret key K to control the tagging and tag checking operations. Then each time U wants to send a message M to V, U should apply the tagging operation to M under the shared secret key K to compute the tag D on M, and convey M and D to V. Finally when V receives M and D, V should apply the tag checking operation to M and D under K to verify the authenticity of M. If the tag checking operation outputs 'valid', V concludes that M is indeed authentic.

Loosely speaking, MAC schemes are designed so that it is hard for an adversary to forge valid message and tag pairs so that the schemes provide data origin authentication and data integrity.

The list of supported MAC schemes at this time is:

HMAC–SHA-1–160 with 20 octet or 160 bit keys
HMAC–SHA-1–80 with 20 octet or 160 bit keys

Both these MAC schemes are specified in IETF RFC 2104 [58] and ANSI X9.71 [5] based on the hash function SHA-1 specified in FIPS 180-1 [30]. Following the notation suggested in [58], here HMAC–*Hash–x* denotes the HMAC function used in conjunction with the hash function *Hash* to produce message

tags of length $x/8$ octets or x bits. Both the supported MAC schemes are designed to be existentially unforgeable in the presence of an adversary capable of launching chosen-message attacks.

(Note that this document does not suggest that other MAC schemes should not be used elsewhere in a system — it merely says that only the MAC schemes listed above should be used to build ECIES.)

For clarity in the remainder of this section, the generic operation of the MAC schemes by U and V is described so that the use of the schemes can be specified precisely later on. The setup procedure is described in Section 3.7.1, the key deployment procedure is specified in Section 3.7.2, the tagging operation is specified in Section 3.7.3, and the tag checking operation is specified in Section 3.7.4.

3.7.1 Scheme Setup

U and V should perform the following setup procedure to use a MAC scheme:

1. U and V should establish which of the supported MAC schemes to use. Let *MAC* denote the MAC scheme chosen, *mackeylen* denote the length in octets of the keys used by the scheme, and *maclen* denote the length in octets of the tags produced by the scheme.

3.7.2 Key Deployment

U and V should perform the following key deployment procedure to use the MAC scheme:

1. U and V should establish a shared secret key K of length *mackeylen* octets. K should be chosen randomly or pseudorandomly.

3.7.3 Tagging Operation

U should tag messages to send to V using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: An octet string M which is the data to be tagged.

Output: An octet string D of length *maclen* octets which is the tag on M, or 'invalid'.

Actions: Compute the tag D on M as follows:

1. Convert M to a bit string $\overline{M}$ and K to a bit string $\overline{K}$ using the conversion routine specified in Section 2.3.2.

2. Calculate the tag $\overline{D}$ on $\overline{M}$ using the selected MAC scheme under the shared secret key $\overline{K}$:

$$\overline{D} = MAC_{\overline{K}}(\overline{M}).$$

   If the MAC scheme outputs 'invalid', output 'invalid' and stop.

3. Convert $\overline{D}$ to an octet string D using the conversion routine specified in Section 2.3.1
4. Output the octet string D of length *maclen* octets.

3.7.4 Tag Checking Operation

V should check the authenticity of tagged messages from U using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The input to the tag checking operation is:

1. An octet string M which is the message.
2. An octet string D which is the purported tag on M.

Output: An indication of whether the tagged message is valid or not — either 'valid' or 'invalid'.

Actions: Check the tag D on M as follows:

1. Convert M to a bit string $\overline{M}$, D to a bit string $\overline{D}$, and K to a bit string $\overline{K}$ using the conversion routine specified in Section 2.3.2.
2. Calculate the tag $\overline{D'}$ on $\overline{M}$ using the selected MAC scheme under the shared secret key $\overline{K}$:

$$\overline{D'} = MAC_{\overline{K}}(\overline{M}).$$

   If the MAC scheme outputs 'invalid', output 'invalid' and stop.
3. Compare $\overline{D'}$ and $\overline{D}$. If $\overline{D'} = \overline{D}$, output 'valid', and if $\overline{D'} \neq \overline{D}$, output 'invalid'.

3.8 Symmetric Encryption Schemes

This section specifies the symmetric encryption schemes supported in this document.

The symmetric encryption schemes will be used by the Elliptic Curve Integrated Encryption Scheme specified in Section 5.1.

Symmetric encryption schemes are designed to be used by two entities — a sender U and a recipient V — when U wants to send a message M to V confidentially, and V wants to recover M.

Here symmetric encryption schemes are described in terms of an encryption operation, a decryption operation, and associated setup and key deployment procedures. U and V should use the scheme as follows when they want to communicate. First U and V should use the setup and key deployment procedures to establish which options to use the scheme with, and to create a shared secret key K to control the encryption and decryption operations. Then each time U wants to send a message M to V, U should apply the encryption operation to M under the shared secret key K to compute the encryption or ciphertext C of M,

and convey C to V. Finally when V receives C, V should apply the decryption operation to C under K to recover the message M.

Loosely speaking, symmetric encryption schemes are designed so that it is hard for an adversary to recover information about messages (other than their length) from their ciphertexts. Thus they provide data confidentiality.

The list of supported symmetric encryption schemes at this time is:

3-key TDES in CBC mode
XOR encryption scheme

(Although only the two encryption schemes above are supported at this time, it is planned that support for AES will be added as AES passes through the standardization procedures of ANSI and NIST.)

(Note that the XOR encryption scheme should be used with care ... here it is recommended specifically for use with the elliptic curve integrated encryption scheme. Implementors considering its use in other circumstances should be aware that it is only designed to provide semantic security when used to encrypt a single message in the presence of adversaries capable of launching only passive attacks.)

3-key TDES in CBC mode is specified in ANSI X9.52 [2]. Here it is considered to use shared secret keys of length 24 octets or 192 bits — which are split up into 3 subkeys K_1, K_2, and K_3 by interpreting the leftmost 8 octets or 64 bits as K_1, the middle 8 octets or 64 bits as K_2, and the rightmost 8 octets or 64 bits as K_3, and replacing the appropriate bits of K_1, K_2, and K_3 with parity bits. Furthermore here the 8 octet or 64 bit IV for TDES in CBC mode should always take the value 0000000000000000_{16}.

The XOR encryption scheme is the simple encryption scheme in which encryption consists of XORing the key and the message, and decryption consists of XORing the key and the ciphertext to recover the message. The XOR scheme is commonly used either with truly random keys when it is known as the ‘one-time pad’, or with pseudorandom keys as a component in the construction of stream ciphers. The XOR encryption scheme uses keys which are the same length as the message to be encrypted or the ciphertext to be decrypted.

3-key TDES in CBC mode is designed to provide semantic security in the presence of adversaries launching chosen-plaintext and chosen-ciphertext attacks. The XOR encryption scheme is designed to provide semantic security when used to encrypt a single message in the presence of adversaries capable of launching only passive attacks. (Although this limits use of the XOR encryption scheme in general, it is sufficient for the purposes of building ECIES. The same is true of TDES with a fixed IV.)

(Note that this document does not suggest that other symmetric encryption schemes should not be used elsewhere in a system — it merely says that only the symmetric encryption schemes listed above should be used to build ECIES.)

For clarity in the remainder of this section, the generic operation of the symmetric encryption schemes by U and V is described so that the use of the schemes can be specified precisely later on. The setup procedure is described in Section 3.8.1, the key deployment procedure is specified in Section 3.8.2, the encryption operation is specified in Section 3.8.3, and the decryption operation is specified in Section 3.8.4.

3.8.1 Scheme Setup

U and *V* should perform the following setup procedure to use a symmetric encryption scheme:

1. *U* and *V* should establish which of the supported symmetric encryption schemes to use. Let *ENC* denote the encryption scheme chosen, and *enckeylen* denote the length in octets of the keys used by the scheme.

3.8.2 Key Deployment

U and *V* should perform the following key deployment procedure to use the symmetric encryption scheme:

1. *U* and *V* should establish a shared secret key *K* of length *enckeylen* octets. *K* should be chosen randomly or pseudorandomly.

3.8.3 Encryption Operation

U should encrypt messages to send to *V* using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: An octet string *M* which is the data to be encrypted.

Output: An octet string *C* which is the ciphertext corresponding to *M*, or 'invalid'.

Actions: Compute the ciphertext *C* as follows:

1. Convert *M* to a bit string $\overline{M}$ and *K* to a bit string $\overline{K}$ using the conversion routine specified in Section 2.3.2.
2. Calculate the encryption $\overline{C}$ of $\overline{M}$ using the encryption operation of the selected symmetric encryption scheme under the shared secret key $\overline{K}$. If the encryption operation outputs 'invalid', output 'invalid' and stop.
3. Convert $\overline{C}$ to an octet string *C* using the conversion routine specified in Section 2.3.1.
4. Output the octet string *C*.

3.8.4 Decryption Operation

V should decrypt ciphertext from *U* using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: An octet string *C* which is the ciphertext.

Output: An octet string *M* which is the decryption of *C*, or 'invalid'.

Actions: Decrypt *C* as follows:

1. Convert *C* to a bit string $\overline{C}$ and *K* to a bit string $\overline{K}$ using the conversion routine specified in Section 2.3.2.
2. Calculate the decryption $\overline{M}$ of $\overline{C}$ using the decryption operation of the selected symmetric encryption scheme under the shared secret key $\overline{K}$. If the decryption operation outputs 'invalid', output 'invalid' and stop.
3. Convert $\overline{M}$ to an octet string *M* using the conversion routine specified in Section 2.3.1.
4. Output the octet string *M*.

4 Signature Schemes

This section specifies the signature schemes based on ECC supported in this document.

Signature schemes are designed to be used by two entities — a signer U and a verifier V — when U wants to send a message M in an authentic manner and V wants to verify the authenticity of M.

Here signature schemes are described in terms of a signing operation, a verifying operation, and associated setup and key deployment procedures. U and V should use the schemes as follows when they want to communicate. First U and V should use the setup procedure to establish which options to use the scheme with, then U should use the key deployment procedure to select a key pair and V should obtain U's public key — U will use the key pair to control the signing operation, and V will use the public key to control the verifying operation. Then each time U wants to send a message M, U should apply the signing operation to M under its key pair to obtain a signature S on M, form a signed message from M and S, and convey the signed message to V. Finally when V receives the signed message, V should apply the verifying operation to the signed message under U's public key to verify its authenticity. If the verifying operation outputs 'valid', V concludes the signed message is indeed authentic.

There are two types of signature schemes, depending on the form of the signed message U must convey to V: signature schemes with appendix in which U must convey both M and S to V, and signature schemes with message recovery in which M can be recovered from S, so U need convey only S to V.

Loosely speaking signature schemes are designed so that it is hard for an adversary who does not know U's secret key to forge valid signed messages so that the schemes provide data origin authentication, data integrity, and non-repudiation.

The only signature scheme supported at this time is the Elliptic Curve Digital Signature Algorithm (ECDSA). ECDSA is specified in Section 4.1.

See Appendix B for a commentary on the contents on this section, including implementation discussion, security discussion, and references.

4.1 Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a signature scheme with appendix based on ECC. It is designed to be existentially unforgeable, even in the presence of an adversary capable of launching chosen-message attacks.

The setup procedure for ECDSA is specified in Section 4.1.1, the key deployment procedure is specified in Section 4.1.2, the signing operation is specified in Section 4.1.3, and the verifying operation is specified in Section 4.1.4.

4.1.1 Scheme Setup

U and V should perform the following setup procedure to prepare to use ECDSA:

1. U should establish which of the hash functions supported in Section 3.5 to use when generating signatures. Let *Hash* denote the hash function chosen, and *hashlen* denote the length in octets of the hash values produced using *Hash*.

2. U should establish elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$ at the desired security level. The elliptic curve domain parameters T should be generated using the primitive specified in Section 3.1.1.1 or the primitive specified in Section 3.1.2.1. U should receive an assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.1.1.2 or Section 3.1.2.2.

3. V should obtain in an authentic manner the hash function *Hash* and elliptic curve domain parameters T established by U.

V may also wish to receive an assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.1.1.2 or Section 3.1.2.2.

4.1.2 Key Deployment

U and V should perform the following key deployment procedure to prepare to use ECDSA:

1. U should establish an elliptic curve key pair (d_U, Q_U) associated with T to use with the signature scheme. The key pair should be generated using the primitive specified in Section 3.2.1.

2. V should obtain in an authentic manner the elliptic curve public key Q_U selected by U.

V may wish to receive an assurance that the elliptic curve public key Q_U is valid using one of the methods specified in Section 3.2.2.

4.1.3 Signing Operation

U should sign messages using ECDSA using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The signing operation takes as input an octet string M which is the message to be signed.

Output: A signature $S = (r, s)$ on M consisting of a pair of integers r and s, or 'invalid'.

Actions: Generate a signature S on M as follows:

1. Select an ephemeral elliptic curve key pair (k, R) with $R = (x_R, y_R)$ associated with the elliptic curve domain parameters T established during the setup procedure using the key pair generation primitive specified in Section 3.2.1.

2. Convert the field element x_R to an integer $\overline{x_R}$ using the conversion routine specified in Section 2.3.9.

3. Set $r \equiv \overline{x_R} \pmod{n}$. If $r = 0$, return to step 1.

4. Use the hash function selected during the setup procedure to compute the hash value:

$$H = Hash(M)$$

of length *hashlen* octets as specified in Section 3.5. If the hash function outputs 'invalid', output 'invalid' and stop.

5. Derive an integer e from H as follows:

   5.1. Convert the octet string H to a bit string $\overline{H}$ using the conversion routine specified in Section 2.3.2.

   5.2. Set $\overline{E} = \overline{H}$ if $\lceil \log_2 n \rceil \geq 8(hashlen)$, and set $\overline{E}$ equal to the leftmost $\lceil \log_2 n \rceil$ bits of $\overline{H}$ if $\lceil \log_2 n \rceil < 8(hashlen)$.

   5.3. Convert the bit string $\overline{E}$ to an octet string E using the conversion routine specified in Section 2.3.1.

   5.4. Convert the octet string E to an integer e using the conversion routine specified in Section 2.3.8.

6. Compute:

$$s \equiv k^{-1}.(e + r.d_U) \pmod{n}.$$

If $s = 0$, return to step 1.

7. Output $S = (r, s)$.

4.1.4 Verifying Operation

V should verify signed messages from U using ECDSA using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The verifying operation takes as input:

1. An octet string M which is the message.

2. U's purported signature $S = (r, s)$ on M.

Output: An indication of whether the purported signature on M is valid or not — either 'valid' or 'invalid'.

Actions: Verify the purported signature S on M as follows:

1. If r and s are not both integers in the interval $[1, n-1]$, output 'invalid' and stop.

2. Use the hash function established during the setup procedure to compute the hash value:

$$H = Hash(M)$$

of length *hashlen* octets as specified in Section 3.5. If the hash function outputs 'invalid', output 'invalid' and stop.

3. Derive an integer e from H as follows:

   3.1. Convert the octet string H to a bit string $\overline{H}$ using the conversion routine specified in Section 2.3.2.

   3.2. Set $\overline{E} = \overline{H}$ if $\lceil \log_2 n \rceil \geq 8(hashlen)$, and set $\overline{E}$ equal to the leftmost $\lceil \log_2 n \rceil$ bits of $\overline{H}$ if $\lceil \log_2 n \rceil < 8(hashlen)$.

   3.3. Convert the bit string $\overline{E}$ to an octet string E using the conversion routine specified in Section 2.3.1.

   3.4. Convert the octet string E to an integer e using the conversion routine specified in Section 2.3.8.

4. Compute:

$$u_1 \equiv e.s^{-1} \pmod{n} \quad \text{and} \quad u_2 \equiv r.s^{-1} \pmod{n}.$$

5. Compute:

$$R = (x_R, y_R) = u_1 G + u_2 Q_U.$$

If $R = O$, output 'invalid' and stop.

6. Convert the field element x_R to an integer $\overline{x_R}$ using the conversion routine specified in Section 2.3.9.

7. Set $v \equiv \overline{x_R} \pmod{n}$.

8. Compare v and r — if $v = r$, output 'valid', and if $v \neq r$, output 'invalid'.

5 Encryption Schemes

This section specifies the public-key encryption schemes based on ECC supported in this document.

Public-key encryption schemes are designed to be used by two entities — a sender U and a recipient V — when U wants to send a message M to V confidentially, and V wants to recover M.

Here public-key encryption schemes are described in terms of an encryption operation, a decryption operation, and associated setup and key deployment procedures. U and V should use the scheme as follows when they want to communicate. First U and V should use the setup procedure to establish which options to use the scheme with, then V should use the key deployment procedure to select a key pair and U should obtain V's public key — U will use V's public key to control the encryption procedure, and V will use its key pair to control the decryption operation. Then each time U wants to send a message M to V, U should apply the encryption operation to M under V's public key to compute an encryption or ciphertext C of M, and convey C to V. Finally when V receives C, V should apply the decryption operation to C under its key pair to recover the message M.

Loosely speaking public-key encryption schemes are designed so that it is hard for an adversary who does not possess V's secret key to recover information about messages (other than their length) from their ciphertexts. Thus the schemes provide data confidentiality.

The public-key encryption schemes specified in this section may be used to encrypt messages of any kind. They may be used to transport keying data from U to V, or to encrypt information data directly. This flexibility allows the schemes to be applied in a broad range of cryptographic systems. Nonetheless it is envisioned that the majority of applications will apply the schemes for key transport, and subsequently use the transported key in conjunction with a symmetric bulk encryption scheme to encrypt information data. This is the traditional usage for public-key encryption schemes.

The only public-key encryption scheme supported at this time is the Elliptic Curve Integrated Encryption Scheme (ECIES). ECIES is specified in Section 5.1.

See Appendix B for a commentary on the contents on this section, including implementation discussion, security discussion, and references.

5.1 Elliptic Curve Integrated Encryption Scheme

The Elliptic Curve Integrated Encryption Scheme (ECIES) is a public-key encryption scheme based on ECC. It is designed to be semantically secure in the presence of an adversary capable of launching chosen-plaintext and chosen-ciphertext attacks.

(Note that the Elliptic Curve Integrated Encryption Scheme has a complex naming history. It is occasionally known instead as the Elliptic Curve Augmented Encryption Scheme or simply the Elliptic Curve Encryption Scheme.)

The setup procedure for ECIES is specified in Section 5.1.1, the key deployment procedure is specified in Section 5.1.2, the encryption operation is specified in Section 5.1.3, and the decryption operation is specified in Section 5.1.4.

5.1.1 Scheme Setup

U and *V* should perform the following setup procedure to prepare to use ECIES:

1. *V* should establish which of the key derivation functions supported in Section 3.6 to use, and select any options involved in the operation of the key derivation function. Let *KDF* denote the key derivation function chosen. (In this edition the only possibility is ANSI-X9.63-KDF with the option SHA-1.)

2. *V* should establish which of the MAC schemes supported in Section 3.7 to use, and select any options involved in the operation of the MAC scheme. Let *MAC* denote the MAC scheme chosen, *mackeylen* denote the length in octets of the keys used by *MAC*, and *maclen* denote the length in octets of tags produced by *MAC*.

3. *V* should establish which of the symmetric encryption schemes supported in Section 3.8 to use, and select any options involved in the operation of the encryption scheme. Let *ENC* denote the encryption scheme chosen, and *enckeylen* denote the length in octets of the keys used by *ENC*.

4. *V* should establish whether to use the 'standard' elliptic curve Diffie-Hellman primitive specified in Section 3.3.1, or the elliptic curve cofactor Diffie-Hellman primitive specified in Section 3.3.2.

5. *V* should establish elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$ at the desired security level. The elliptic curve domain parameters T should be generated using the primitive specified in Section 3.1.1.1 or the primitive specified in Section 3.1.2.1. *V* should receive an assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.1.1.2 or Section 3.1.2.2.

6. *U* should obtain in an authentic manner the selections made by *V* — the key derivation function *KDF*, the MAC scheme *MAC*, the symmetric encryption scheme *ENC*, the elliptic curve domain parameters T, and an indication whether to use the 'standard' elliptic curve Diffie-Hellman primitive or the cofactor Diffie-Hellman. *U* should also receive an assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.1.1.2 or Section 3.1.2.2.

7. *U* should establish whether or not to represent elliptic curve points using point compression.

5.1.2 Key Deployment

U and *V* should perform the following key deployment procedure to prepare to use ECIES:

1. *V* should establish an elliptic curve key pair (d_V, Q_V) associated with the elliptic curve domain parameters T established during the setup procedure. The key pair should be generated using the primitive specified in Section 3.2.1.

2. U should obtain in an authentic manner the elliptic curve public key Q_V selected by V. If the 'standard' elliptic curve Diffie-Hellman primitive is being used, U should receive an assurance that Q_V is valid using one of the methods specified in Section 3.2.2, and if the elliptic curve cofactor Diffie-Hellman primitive is being used, U should receive an assurance that Q_V is at least partially valid using one of the methods specified in Section 3.2.2 or Section 3.2.3.

5.1.3 Encryption Operation

U should encrypt messages using ECIES using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The input to the encryption operation is:

1. An octet string M which is the message to be encrypted.
2. (Optional) Two octet strings $SharedInfo_1$ and $SharedInfo_2$ which consist of some data shared by U and V.

Output: An octet string C which is an encryption of M, or 'invalid'.

Actions: Encrypt M as follows:

1. Select an ephemeral elliptic curve key pair (k, R) with $R = (x_R, y_R)$ associated with the elliptic curve domain parameters T established during the setup procedure. Generate the key pair using the key pair generation primitive specified in Section 3.2.1.
2. Convert R to an octet string $\overline{R}$ using the conversion routine specified in Section 2.3.3. Decide whether or not to represent R using point compression according to the convention established during the setup procedure.
3. Use one of the Diffie-Hellman primitives specified in Section 3.3 to derive a shared secret field element $z \in \mathbb{F}_q$ from the ephemeral secret key k and V's public key Q_V obtained during the key deployment procedure. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop. Decide whether to use the 'standard' elliptic curve Diffie-Hellman primitive or the elliptic curve cofactor Diffie-Hellman primitive according to the convention established during the setup procedure.
4. Convert $z \in \mathbb{F}_q$ to an octet string Z using the conversion routine specified in Section 2.3.5.
5. Use the key derivation function KDF established during the setup procedure to generate keying data K of length $enckeylen + mackeylen$ octets from Z and $[SharedInfo_1]$. If the key derivation function outputs 'invalid', output 'invalid' and stop.
6. Parse the leftmost $enckeylen$ octets of K as an encryption key EK and the rightmost $mackeylen$ octets of K as a MAC key MK.

7. Use the encryption operation of the symmetric encryption scheme *ENC* established during the setup procedure to encrypt M under EK as ciphertext EM. If the encryption scheme outputs 'invalid', output 'invalid' and stop.

8. Use the tagging operation of the MAC scheme *MAC* established during the setup procedure to compute the tag D on $EM \,||\, [SharedInfo_2]$ under MK. If the MAC scheme outputs 'invalid', output 'invalid' and stop.

9. Output $C = \overline{R} \,||\, EM \,||\, D$.

5.1.4 Decryption Operation

V should decrypt ciphertext using ECIES using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The input to the decryption operation is:

1. An octet string C which is the ciphertext.

2. (Optional) Two octet strings $SharedInfo_1$ and $SharedInfo_2$ which consist of some data shared by U and V.

Output: An octet string M which is the decryption of C, or 'invalid'.

Actions: Decrypt C as follows:

1. If the leftmost octet of C is 02_{16} or 03_{16}, parse the leftmost $\lceil \log_2 q \rceil + 1$ octets of C as an octet string $\overline{R}$, the rightmost *maclen* octets of C as an octet string D, and the remaining octets of C as an octet string EM. If the leftmost octet of C is 04_{16}, parse the leftmost $2\lceil \log_2 q \rceil + 1$ octets of C as an octet string $\overline{R}$, the rightmost *maclen* octets of C as an octet string D, and the remaining octets of C as an octet string EM. If the leftmost octet of C is not 02_{16}, 03_{16}, or 04_{16}, output 'invalid' and stop.

2. Convert the octet string $\overline{R}$ to an elliptic curve point $R = (x_R, y_R)$ associated with the elliptic curve domain parameters T established during the setup procedure using the conversion routine specified in Section 2.3.4. If the conversion routine outputs 'invalid', output 'invalid' and stop.

3. If the 'standard' elliptic curve Diffie-Hellman primitive is being used, receive an assurance that R is a valid elliptic curve public key using one of the methods specified in Section 3.2.2. If the elliptic curve cofactor Diffie-Hellman primitive is being used, receive an assurance that R is at least a partially valid elliptic curve public key using one of the methods specified in Section 3.2.2 Section 3.2.3. If an appropriate assurance is not obtained, output 'invalid' and stop.

4. Use one of the Diffie-Hellman primitives specified in Section 3.3 to derive a shared secret field element $z \in \mathbb{F}_q$ from V's secret key d_V established during the key deployment procedure and the public key R. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop. Decide

whether to use the 'standard' elliptic curve Diffie-Hellman primitive or the elliptic curve cofactor Diffie-Hellman primitive according to the convention established during the setup procedure.

5. Convert $z \in \mathbb{F}_q$ to an octet string Z using the conversion routine specified in Section 2.3.5.

6. Use the key derivation function *KDF* established during the setup procedure to generate keying data K of length $enckeylen + mackeylen$ octets from Z and $[SharedInfo_1]$. If the key derivation function outputs 'invalid', output 'invalid' and stop.

7. Parse the leftmost *enckeylen* octets of K as an encryption key EK and the rightmost *mackeylen* octets of K as a MAC key MK.

8. Use the tag checking operation of the MAC scheme *MAC* established during the setup procedure to check that D is the tag on $EM \,||\, [SharedInfo_2]$ under MK. If the MAC scheme outputs 'invalid', output 'invalid' and stop.

9. Use the decryption operation of the symmetric encryption scheme *ENC* established during the setup procedure to decrypt EM under EK as M. If the encryption scheme outputs 'invalid', output 'invalid' and stop.

10. Output M.

Note that when implementing the above decryption operation on a constrained device, it may be desirable to perform the symmetric decryption operation (step 9) before the tag checking operation (step 8). This variant is allowed. However implementers of this variant should guard against the possibility that the output of step 9 is available to attackers regardless of the output of step 8.

6 Key Agreement Schemes

This section specifies the key agreement schemes based on ECC supported in this document.

Key agreement schemes are designed to be used by two entities — an entity *U* and an entity *V* — when *U* and *V* want to establish shared keying data that they can later use to control the operation of a symmetric cryptographic scheme.

Here key agreement schemes are described in terms of a key agreement operation, and associated setup and key deployment procedures. *U* and *V* should use the schemes as follows when they want to establish keying data. First *U* and *V* should use the setup procedure to establish which options to use the scheme with, then they should use the key deployment procedure to select key pairs and exchange public keys. Finally when *U* and *V* want to establish keying data they should simultaneously use the key agreement operation. Provided *U* and *V* operate the key agreement operation with corresponding keys as inputs, they will obtain the same keying data.

Note that this document does not address how specific keys should be derived from keying data established using a key agreement scheme. This detail is left to be determined on an application by application basis. Some applications may wish simply to use the keying data directly as a key, others may wish to split the keying data into more than one key, and others may wish to process the keying data to exclude weak keys.

Key agreement schemes are designed to meet a wide variety of security goals depending on how they are applied — security goals that the schemes described here are designed to provide include unilateral implicit key authentication, mutual implicit key authentication, known-key security, and forward secrecy, in the presence of adversaries capable of launching both passive and active attacks.

Two key agreement schemes are supported at this time: the elliptic curve Diffie-Hellman scheme, and the elliptic curve MQV scheme. The elliptic curve Diffie-Hellman scheme is specified in Section 6.1, and the elliptic curve MQV scheme is specified in Section 6.2.

See Appendix B for a commentary on the contents on this section, including implementation discussion, security discussion, and references.

6.1 Elliptic Curve Diffie-Hellman Scheme

The elliptic curve Diffie-Hellman scheme is a key agreement scheme based on ECC. It is designed to provide a variety of security goals depending on its application — goals it can provide include unilateral implicit key authentication, mutual implicit key authentication, known-key security, and forward secrecy — depending on issues like whether or not public keys are exchanged in an authentic manner, and whether key pairs are ephemeral or static. See Appendix B for a further discussion.

The setup procedure for the elliptic curve Diffie-Hellman scheme is specified in Section 6.1.1, the key deployment procedure is specified in Section 6.1.2, and the key agreement operation is specified in Section 6.1.3.

6.1.1 Scheme Setup

U and V should perform the following setup procedure to prepare to use the elliptic curve Diffie-Hellman scheme:

1. U and V should establish which of the key derivation functions supported in Section 3.6 to use, and select any options involved in the operation of the key derivation function. Let *KDF* denote the key derivation function chosen.

2. U and V should establish whether to use the 'standard' elliptic curve Diffie-Hellman primitive specified in Section 3.3.1, or the elliptic curve cofactor Diffie-Hellman primitive specified in Section 3.3.2.

3. U and V should establish at the desired security level elliptic curve domain parameters $T = (p,a,b,G,n,h)$ or $(m,f(x),a,b,G,n,h)$. The elliptic curve domain parameters T should be generated using the primitive specified in Section 3.1.1.1 or the primitive specified in Section 3.1.2.1. Both U and V should receive an assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.1.1.2 or Section 3.1.2.2.

6.1.2 Key Deployment

U and V should perform the following key deployment procedure to prepare to use the elliptic curve Diffie-Hellman scheme:

1. U should establish an elliptic curve key pair (d_U, Q_U) associated with the elliptic curve domain parameters T established during the setup procedure. The key pair should be generated using the primitive specified in Section 3.2.1.

2. V should establish an elliptic curve key pair (d_V, Q_V) associated with the elliptic curve domain parameters T established during the setup procedure. The key pair should be generated using the primitive specified in Section 3.2.1.

3. U and V should exchange their public keys Q_U and Q_V.

4. If the 'standard' elliptic curve Diffie-Hellman primitive is being used, U should receive an assurance that Q_V is valid using one of the methods specified in Section 3.2.2, and if the elliptic curve cofactor Diffie-Hellman primitive is being used, U should receive an assurance that Q_V is at least partially valid using one of the methods specified in Section 3.2.2 or Section 3.2.3.

5. If the 'standard' elliptic curve Diffie-Hellman primitive is being used, V should receive an assurance that Q_U is valid using one of the methods specified in Section 3.2.2, and if the elliptic curve cofactor Diffie-Hellman primitive is being used, V should receive an assurance that Q_U is at least partially valid using one of the methods specified in Section 3.2.2 or Section 3.2.3.

6.1.3 Key Agreement Operation

U and V should perform the key agreement operation described in this section to establish keying data using the elliptic curve Diffie-Hellman scheme. For clarity U's use of the operation is described. V's use of the operation is analogous, but with the roles of U and V reversed.

U should establish keying data with V using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The input to the key agreement operation is:

1. An integer *keydatalen* which is the number of octets of keying data required.
2. (Optional) An octet string $SharedInfo$ which consists of some data shared by U and V.

Output: An octet string K which is the keying data of length *keydatalen* octets, or 'invalid'.

Actions: Establish keying data as follows:

1. Use one of the Diffie-Hellman primitives specified in Section 3.3 to derive a shared secret field element $z \in \mathbb{F}_q$ from U's secret key d_U established during the key deployment procedure and V's public key Q_V obtained during the key deployment procedure. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop. Decide whether to use the 'standard' elliptic curve Diffie-Hellman primitive or the elliptic curve cofactor Diffie-Hellman primitive according to the convention established during the setup procedure.
2. Convert $z \in \mathbb{F}_q$ to an octet string Z using the conversion routine specified in Section 2.3.5.
3. Use the key derivation function KDF established during the setup procedure to generate keying data K of length *keydatalen* octets from Z and $[SharedInfo]$. If the key derivation function outputs 'invalid', output 'invalid' and stop.
4. Output K.

6.2 Elliptic Curve MQV Scheme

The elliptic curve MQV scheme is a key agreement scheme based on ECC. It is designed to provide a variety of security goals depending on its application — goals it can provide include mutual implicit key authentication, known-key security, and forward secrecy — depending on issues like whether or not U and V both contribute ephemeral key pairs. See Appendix B for a further discussion.

The setup procedure for the elliptic curve MQV scheme is specified in Section 6.2.1, the key deployment procedure is specified in Section 6.2.2, and the key agreement operation is specified in Section 6.2.3.

6.2.1 Scheme Setup

U and V should perform the following setup procedure to prepare to use the elliptic curve MQV scheme:

1. U and V should establish which of the key derivation functions supported in Section 3.6 to use, and select any options involved in the operation of the key derivation function. Let *KDF* denote the key derivation function chosen.

2. U and V should establish at the desired security level elliptic curve domain parameters $T = (p,a,b,G,n,h)$ or $(m,f(x),a,b,G,n,h)$. The elliptic curve domain parameters T should be generated using the primitive specified in Section 3.1.1.1 or the primitive specified in Section 3.1.2.1. Both U and V should receive an assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.1.1.2 or Section 3.1.2.2.

6.2.2 Key Deployment

U and V should perform the following key deployment procedure to prepare to use the elliptic curve MQV scheme:

1. U should establish two elliptic curve key pairs $(d_{1,U},Q_{1,U})$ and $(d_{2,U},Q_{2,U})$ associated with the elliptic curve domain parameters T established during the setup procedure. The key pairs should both be generated using the primitive specified in Section 3.2.1.

2. V should establish two elliptic curve key pairs $(d_{1,V},Q_{1,V})$ and $(d_{2,V},Q_{2,V})$ associated with the elliptic curve domain parameters T established during the setup procedure. The key pairs should both be generated using the primitive specified in Section 3.2.1.

3. U should obtain in an authentic manner the first elliptic curve public key $Q_{1,V}$ selected by V. U should receive an assurance that $Q_{1,V}$ is valid using one of the methods specified in Section 3.2.2.

4. V should obtain in an authentic manner the first elliptic curve public key $Q_{1,U}$ selected by U. V should receive an assurance that $Q_{1,U}$ is valid using one of the methods specified in Section 3.2.2.

5. U and V should exchange their second public keys $Q_{2,U}$ and $Q_{2,V}$.

6. U should receive an assurance that $Q_{2,V}$ is at least partially valid using one of the methods specified in Section 3.2.2 or Section 3.2.3.

7. V should receive an assurance that $Q_{2,U}$ is at least partially valid using one of the methods specified in Section 3.2.2 or Section 3.2.3.

6.2.3 Key Agreement Operation

U and V should perform the key agreement operation described in this section to establish keying data using the elliptic curve MQV scheme. For clarity U's use of the operation is described. V's use of the operation is analogous, but with the roles of U and V reversed.

U should establish keying data with V using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The input to the key agreement operation is:

1. An integer *keydatalen* which is the number of octets of keying data required.
2. (Optional) An octet string $SharedInfo$ which consists of some data shared by U and V.

Output: An octet string K which is the keying data of length *keydatalen* octets, or 'invalid'.

Actions: Establish keying data as follows:

1. Use the elliptic curve MQV primitive specified in Section 3.4 to derive a shared secret field element $z \in \mathbb{F}_q$ from U's key pairs $(d_{1,U}, Q_{1,U})$ and $(d_{2,U}, Q_{2,U})$ established during the key deployment procedure and V's public keys $Q_{1,V}$ and $Q_{2,V}$ obtained during the key deployment procedure. If the MQV primitive outputs 'invalid', output 'invalid' and stop.
2. Convert $z \in \mathbb{F}_q$ to an octet string Z using the conversion routine specified in Section 2.3.5.
3. Use the key derivation function KDF established during the setup procedure to generate keying data K of length *keydatalen* octets from Z and $[SharedInfo]$. If the key derivation function outputs 'invalid', output 'invalid' and stop.
4. Output K.

A Glossary

This section supplies a glossary to the terms and notation used in this document.

The section is organized as follows. Section A.1 lists the terms used in this document, Section A.2 lists the acronyms used, and Section A.3 specifies the notation used.

A.1 Terms

Terms used in this document include:

active attack	The ability of an adversary of a cryptographic scheme to subvert communications between entities by deleting, injecting, substituting, or generally subverting messages in any way.
addition rule	An addition rule describes the addition of two elliptic curve points P_1 and P_2 to produce a third elliptic curve point P_3. See Section 2.2.
base point	A distinguished point G on an elliptic curve.
binary polynomial	A polynomial whose coefficients are either 0's or 1's.
bit string	A bit string is an ordered sequence of 0's and 1's.
certificate	The public key and identity of an entity together with some other information, rendered unforgeable by signing this information with the secret key of a Certification Authority.
Certification Authority	A Center trusted by one or more entities to create and assign certificates.
characteristic 2 finite field	A finite field containing 2^m elements, where $m \geq 1$ is an integer.
chosen-ciphertext attack	The ability of an adversary of an encryption scheme to obtain the decryptions of ciphertexts of its choice in an attempt to compromise the scheme.
chosen-message attack	The ability of an adversary of a signature scheme to obtain signatures on messages of its choice in an attempt to compromise the scheme. Or, similarly the ability of an adversary of a MAC scheme to obtain tags on messages of its choice in an attempt to compromise the scheme.
chosen-plaintext attack	The ability of an adversary of an encryption scheme to obtain the encryptions of plaintexts of its choice in an attempt to compromise the scheme.
ciphertext	The result of applying an encryption operation to a message.

cryptographic scheme	A cryptographic scheme consists of an unambiguous specification of a set of operations capable of providing a security service when properly implemented and maintained.
data confidentiality	The assurance that data is unintelligible to unauthorized parties.
data integrity	The assurance that data has not been modified by unauthorized parties.
data origin authentication	The assurance that the purported origin of data is correct.
elliptic curve	An elliptic curve over $\mathbb{F}_q$ is a set of points which satisfy a certain equation specified by two parameters, a and b, which are elements of the field $\mathbb{F}_q$. See Section 2.2.
elliptic curve domain parameters	Elliptic curve domain parameters are comprised of a field size q, a reduction polynomial $f(x)$ in the case $q = 2^m$, two elements a, b in $\mathbb{F}_q$ which define an elliptic curve E over $\mathbb{F}_q$, a point G of prime order in $E(\mathbb{F}_q)$, the order n of G, and the cofactor h. See Section 3.1.
elliptic curve key pair	Given particular elliptic curve domain parameters, an elliptic curve key pair (d, Q) consists of an elliptic curve secret key d and the corresponding elliptic curve public key Q.
elliptic curve point	If E is an elliptic curve defined over $\mathbb{F}_q$, then an elliptic curve point P is either a pair of field elements (x_P, y_P) (where $x_P, y_P \in \mathbb{F}_q$) such that the values $x = x_P$ and $y = y_P$ satisfy the equation defining E, or a special point O called the point at infinity.
elliptic curve public key	Given particular elliptic curve domain parameters, and an elliptic curve secret key d, the corresponding elliptic curve public key Q is the elliptic curve point $Q = dG$, where G is the base point.
elliptic curve secret key	Given particular elliptic curve domain parameters, an elliptic curve secret key d is an integer in the interval $[1, n-1]$, where n is the prime order of the base point G.
encryption scheme	An encryption scheme is a cryptographic scheme consisting of an encryption operation and a decryption operation which is capable of providing data confidentiality.
entity	A party involved in the operation of a cryptographic system.
ephemeral	Ephemeral data is relatively short-lived. Usually ephemeral data refers to data specific to a particular execution of a cryptographic scheme.

existentially unforgeable	A signature scheme is existentially unforgeable if it is infeasible for an adversary to forge a signature on any message that has not previously been signed by the scheme's legitimate user. Similarly a MAC scheme is existentially unforgeable if it is infeasible for an adversary to forge the tag on any message that has not previously been tagged by one of the scheme's legitimate users.
forward secrecy	The assurance that a session key established between some parties will not be compromised by the compromise of some of the parties' static secret keys in the future. Also known as perfect forward secrecy.
hash function (cryptographic hash function)	A cryptographic hash function is a function which maps bit strings from a large (possibly very large) domain into a smaller range. The function possesses the following properties: it is computationally infeasible to find any input which maps to a randomly given output, and nobody can exhibit a pair of distinct inputs which map to the same output.
implicit key authentication	A key establishment scheme provides implicit key authentication if only authorized parties are possibly capable of computing any session key.
irreducible binary polynomial	A binary polynomial $f(x)$ is irreducible if it does not factor over $\mathbb{F}_2$ as a product of two or more binary polynomials, each of degree less than the degree of $f(x)$.
key (cryptographic key)	A parameter that determines the execution of a cryptographic operation such as the transformation from plaintext to ciphertext and vice versa, the synchronized generation of keying material, or a digital signature computation or validation.
key agreement scheme	A key agreement scheme is a key establishment scheme in which the keying data established is a function of contributions provided by each party to the scheme in such a way that no party can predetermine the value of the keying data.
key derivation function	A key derivation function is a function which takes as input a shared secret value and outputs keying data suitable for later cryptographic use.
key establishment scheme	A key establishment scheme is a cryptographic scheme which reveals only to its legitimate users keying data suitable for subsequent cryptographic use by cryptographic schemes.
keying data	Data suitable for use as cryptographic keys.

known-key security	The assurance that a session key established by an execution of a key establishment scheme will not be compromised by the compromise of other session keys.
MAC scheme	A MAC scheme is a cryptographic scheme consisting of a message tagging operation and a tag checking operation which is capable of providing data origin authentication and data integrity.
non-repudiation	The assurance that the origin and integrity of data can be proved to a third party.
octet	An octet is a bit string of length 8. An octet is represented by a hexadecimal string of length 2. The first hexadecimal digit represents the four leftmost bits of the octet, and the second hexadecimal digit represents the four rightmost bits of the octet. For example, $9D$ represents the bit string 10011101. An octet also represents an integer in the interval $[0, 255]$. For example, $9D$ represents the integer 157.
octet string	An octet string is an ordered sequence of octets.
order of a curve	The order of an elliptic curve E defined over the field $\mathbb{F}_q$ is the number of points on the elliptic curve E, including O. This is denoted by $\#E(\mathbb{F}_q)$.
order of a point	The order of a point P is the smallest positive integer n such that $nP = O$ (the point at infinity).
partially valid elliptic curve public key	An elliptic curve public key $Q = (x_Q, y_Q)$ is partially valid if the values $x = x_Q$ and $y = y_Q$ satisfy the defining equation of the associated elliptic curve E, but it is not necessarily the case that $Q = dG$ for some d. The elliptic curve public key partial validation primitive in Section 3.2.3.1 checks whether or not an elliptic curve public key is partially valid.
passive attack	The ability of an adversary of a cryptographic scheme merely to observe entities communicating using the scheme.
pentanomial	A binary polynomial of the form $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 \leq k_1 < k_2 < k_3 \leq m-1$.
plaintext	A message to be encrypted; the opposite of ciphertext.
point compression	Point compression allows a point $P = (x_P, y_P)$ to be represented compactly using x_P and a single additional bit $\tilde{y}_P$ derived from x_P and y_P. See Section 2.3.
prime finite field	A finite field containing p elements, where p is an odd prime number.

primitive (cryptographic primitive)	A cryptographic primitive is an operation not by itself capable of providing a security service, but which can be incorporated in a cryptographic scheme.
public key	In a public-key scheme, that key of an entity's key pair which can be publicly known.
public-key cryptographic scheme	A cryptographic scheme in which each operation is controlled by one of two keys; either the public key or the private key. The two keys have the property that, given the public key, it is computationally infeasible to derive the private key. Also known as an asymmetric cryptographic scheme.
reduction polynomial	The irreducible binary polynomial $f(x)$ of degree m that is used to determine a representation of $\mathbb{F}_{2^m}$.
scalar multiplication	If k is a positive integer, then $k \times P$ or kP denotes the point obtained by adding together k copies of the point P. The process of computing kP from P and k is called scalar multiplication.
secret key	In a public-key system, that key of an entity's key pair which must be known only by that entity. Also known as a private key.
semantically secure	An encryption scheme is semantically secure if it is infeasible for an adversary to learn anything from ciphertext about the corresponding plaintext apart from the length of the plaintext.
session key	A key (usually short-lived) established using a key establishment scheme.
shared secret value	An intermediate value in a key establishment scheme from which keying data is derived.
signature scheme	A signature scheme is a cryptographic scheme consisting of a signing operation and a verifying operation which is capable of providing data origin authentication, data integrity, and non-repudiation.
static	Static data is relatively long-lived. Usually static data refers to data common to a number of executions of a cryptographic scheme.
symmetric cryptographic scheme	A cryptographic scheme in which each operation is controlled by the same key.
trinomial	A binary polynomial of the form $x^m + x^k + 1$, where $1 \leq k \leq m-1$.
unknown key-share resilience	The assurance that all the parties who share a session key are aware which parties they share the key with.

valid elliptic curve domain parameters	Elliptic curve domain parameters are valid if they satisfy the arithmetic requirements of elliptic curve domain parameters. Equivalently elliptic curve domain parameters are valid if they have been generated as specified in Section 3.1.1.1 or 3.1.2.1. The elliptic curve domain parameter validation primitives in Sections 3.1.1.2.1 and 3.1.2.2.1 check whether or not elliptic curve domain parameters are valid.
valid elliptic curve public key	An elliptic curve public key $Q = (x_Q, y_Q)$ is valid if it satisfies the arithmetic requirements of an elliptic curve public key — namely that $Q = dG$ for some d in the interval $[1, n-1]$ where G is the base point of the associated elliptic curve domain parameters and n is the order of G. The elliptic curve public key validation primitive in Section 3.2.2.1 checks whether or not an elliptic curve public key is valid.
x-coordinate	The x-coordinate of an elliptic curve point $P = (x_P, y_P)$ is x_P.
y-coordinate	The y-coordinate of an elliptic curve point $P = (x_P, y_P)$ is y_P.

A.2 Acronyms

The acronyms used in this document denote:

AES	Advanced Encryption Standard.
ANSI	American National Standards Institute.
ASN.1	Abstract Syntax Notation One.
CA	Certification Authority.
DES	Data Encryption Standard. See [29].
DSA	Digital Signature Algorithm. See [31].
EC	Elliptic Curve.
ECC	Elliptic Curve Cryptography.
ECDH	Elliptic Curve Diffie-Hellman.
ECDLP	Elliptic Curve Discrete Logarithm Problem.
ECDSA	Elliptic Curve Digital Signature Algorithm. See Section 4.1.
ECIES	Elliptic Curve Integrated Encryption Scheme. See Section 5.1.
ECMQV	Elliptic Curve Menezes-Qu-Vanstone.
FSML	Financial Services Markup Language.

FSTC	Financial Services Technology Consortium.
HMAC	Hash-based Message Authentication Code. See [58].
IEC	International Electrotechnical Commission.
IEEE	Institute of Electrical and Electronics Engineers.
IETF	Internet Engineering Task Force.
ISO	International Organization for Standardization.
MQV	Menezes-Qu-Vanstone. See [60].
NIST	(U.S.) National Institute of Standards and Technology.
PKI	Public Key Infrastructure.
PKIX	Public Key Infrastructure for the Internet.
SHA-1	Secure Hash Algorithm Revision One. See [30].
SSL	Secure Sockets Layer.
TDES	Triple Data Encryption Standard. See [2].
TLS	Transport Layer Security.
WAP	Wireless Application Protocol.
WTLS	Wireless Transport Layer Security.

A.3 Notation

The notation adopted in this document is:

$\|\|X\|\|$	Length in octets of the octet string X.
$[X]$	Indicates that the inclusion of X is optional.
$[x,y]$	The interval of integers between and including x and y.
$\lceil x \rceil$	Ceiling: the smallest integer $\geq x$. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.
$\lfloor x \rfloor$	Floor: the largest integer $\leq x$. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.
$x \bmod n$	The unique remainder r, $0 \leq r \leq n-1$, when x is divided by n. For example, $23 \bmod 7 = 2$.
C	Ciphertext.
d	An EC private key.
E	An elliptic curve over the field $\mathbb{F}_q$ defined by a and b.

$E(\mathbb{F}_q)$	The set of all points on an elliptic curve E defined over $\mathbb{F}_q$ and including the point at infinity O.
$\#E(\mathbb{F}_q)$	If E is defined over $\mathbb{F}_q$, then $\#E(\mathbb{F}_q)$ denotes the number of points on the curve (including the point at infinity O). $\#E(\mathbb{F}_q)$ is called the order of the curve E.
$\mathbb{F}_{2^m}$	The finite field containing 2^m elements, where m is a positive integer.
$\mathbb{F}_p$	The finite field containing p elements, where p is a prime.
$\mathbb{F}_q$	The finite field containing q elements. In this document attention is restricted to the cases that q is an odd prime number (p) or a power of 2 (2^m).
G	A distinguished point on an elliptic curve called the base point or generating point.
$\gcd(x,y)$	The greatest common divisor of positive integers x and y.
h	$h = \#E(\mathbb{F}_q)/n$, where n is the order of the base point G. h is called the co-factor.
k	An EC private key specific to one particular instance of a cryptographic scheme.
K	Keying data.
$\log_2 x$	The logarithm of x to the base 2.
m	The degree of the finite field $\mathbb{F}_{2^m}$.
M	A message.
mod	Modulo.
$\bmod f(x)$	Arithmetic modulo the polynomial $f(x)$.
$\bmod n$	Arithmetic modulo n.
n	The order of the base point G.
O	A special point on an elliptic curve, called the point at infinity. This is the additive identity of the elliptic curve group.
p	An odd prime number.
P	An EC point.
q	The number of elements in the field $\mathbb{F}_q$.
Q	An EC public key.
R	An EC public key specific to one particular instance of a cryptographic scheme.
S	A digital signature.

T	Elliptic curve domain parameters.
U, V	Entities.
$X \| \| Y$	Concatenation of two strings X and Y. X and Y are either both bit strings or both octet strings.
$X \oplus Y$	Bitwise exclusive-or of two bit strings X and Y of the same bit length.
x_P	The x-coordinate of a point P.
y_P	The y-coordinate of a point P.
$\tilde{y}_P$	The representation of the y-coordinate of a point P when point compression is used.
z, or Z	A shared secret value. z denotes a shared secret integer or field element, and Z a shared secret bit string or octet string.
$\mathbb{Z}_p$	The set of integers modulo p, where p is an odd prime number.

Furthermore positional notation is used to indicate the association of a value to a particular entity. For example d_U denotes an EC private key owned by entity U. Occasionally positional notation is also used to indicate a counter value associated with some data, or to indicate the base in which a particular value is being expressed if there is some possibility of ambiguity. For example, $Hash_1$ denotes the value of $Hash_i$ when the counter i has value 1, and 01_{16} denotes that the value 01 is written in hexadecimal.

With the exception of notation that has been well-established in other documents, where possible in this document capital letters are used in variable names that denote bit strings or octet strings, and capital letters are excluded from variable names that denote field elements or integers. For example, d is used to denote the integer that specifies an EC private key, and M is used to denote the octet string to be signed using a signature scheme.

B Commentary

This section provides a commentary on the main body of this document, including implementation discussion, security discussion, and references.

The aim of this section is to supply implementers with relevant guidance. However the section does not attempt to provide exhaustive information but rather focuses on giving basic information and including pointers to references which contain additional material. Furthermore the section concentrates on supplying information specific to ECC rather than providing extensive information on components like SHA-1 and TDES which are specified elsewhere.

The information in this section is current as of September 2000. The information is likely to change over time, and implementers should therefore survey the state-of-the-art at the time of implementation and carry out periodic reviews subsequent to deployment.

This section is organized as follows. Sections B.1 through B.5 respectively provide commentary on Sections 2 through 6 of the main body of this document. Section B.6 supplies information regarding the alignment of this document with other standards efforts which include ECC.

B.1 Commentary on Section 2 - Mathematical Foundations

This section provides a commentary on Section 2 of the main body of this document.

Finite fields and elliptic curves have been studied as mathematical objects for hundreds of years. The body of literature on these structures is vast. Introductions to finite fields can be found in the books of Jungnickel [49], Lidl and Neiderreiter [61], and McEliece [63]. An introduction to elliptic curves can be found in the book of Silverman [81].

Elliptic curves over finite fields were first proposed for use to build cryptographic schemes in 1985 by Koblitz [52] and Miller [68]. Excellent treatments focusing on ECC are contained in Blake, Seroussi, and Smart [12], Koblitz [54], Koblitz, Menezes, and Vanstone [55], and Menezes [64].

The security of all cryptographic schemes based on ECC relies on the elliptic curve discrete logarithm problem or ECDLP. The ECDLP is stated as follows in the case of interest here - namely when the elliptic curve in question has order divisible by a large prime n.

Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, and let $G \in E(\mathbb{F}_q)$ be a point on E of large prime order n. The ECDLP is, given E, G, and a scalar multiple Q of G, to determine an integer l such that $Q = lG$.

No general subexponential algorithms are known for the ECDLP. The best general algorithms known to date are based on the Pollard-ρ method and the Pollard-λ method [76]. The Pollard-ρ method takes about $\sqrt{\pi n/2}$ steps, and the Pollard-λ method takes about $2\sqrt{n}$ steps. Both methods can be parallelized effectively - see [84].

Gallant, Lambert, and Vanstone [35], and Wiener and Zuccherato [89] recently showed that the Pollard-ρ method can be sped up by a factor of $\sqrt{2}$. Thus the expected running time of the Pollard-ρ method with

this speedup is $\sqrt{\pi n/4}$ steps.

They also showed that this speedup can be enhanced when E is an elliptic curve over $\mathbb{F}_{2^{ed}}$ which is defined over $\mathbb{F}_{2^e}$. In this case they show that the Pollard-ρ method can be sped up by a factor of $\sqrt{2d}$. For example, the Koblitz curve $E: y^2+xy=x^3+x^2+1$ over $\mathbb{F}_{2^{163}}$ has the property that $\#E(\mathbb{F}_{2^{163}})=2n$ where n is a 162-bit prime. As a result of the enhancement to the Pollard-ρ method, the ECDLP in $E(\mathbb{F}_{2^{163}})$ can be solved in about 2^{77} steps as opposed to the 2^{81} steps required to solve the ECDLP for a random curve of similar order.

Table 2 below illustrates the difficulty of the ECDLP. It contains estimates in MIPS years of the computing power required to solve the ECDLP on a general curve in software using the improved Pollard-ρ method. To place Table 2 in context, Odlyzko has estimated that 0.1% of the world's computing power working for 1 year will amount to 10^8 MIPS years in 2004, and 10^{10} or 10^{11} MIPS years in 2014 [71]. Table 2 is reproduced from ANSI X9.62 [3]. More details on how the estimates were obtained can be found there.

Size of n (in bits)	$\sqrt{\pi n/4}$	MIPS years
160	2^{80}	8.5×10^{11}
186	2^{93}	7.0×10^{15}
234	2^{117}	1.2×10^{23}
354	2^{177}	1.3×10^{41}
426	2^{213}	9.2×10^{51}

Table 2: Computing power required to solve ECDLP

The difficulty of the ECDLP is further illustrated by van Oorschot and Wiener's 1994 paper [84]. Van Oorschot and Wiener carried out a detailed study of the feasibility of building special-purpose hardware to solve the ECDLP. They estimated that for about $10 million a machine with 325,000 processors could be built that would solve the ECDLP for an elliptic curve E with $n\approx2^{120}$ in about 35 days. Hardware attacks on larger values of n, like $n\approx2^{160}$, appear completely infeasible at this time.

Although no general subexponential algorithms to solve the ECDLP are known, three classes of curves are susceptible to special-purpose algorithms. Firstly, elliptic curves E over $\mathbb{F}_q$ with n dividing q^B-1 for small B are susceptible to the attacks described by Menezes, Okamoto, and Vanstone [65], and Frey and Ruck [32]. These attacks efficiently reduce the ECDLP on these curves to the traditional discrete logarithm problem in a small degree extension of $\mathbb{F}_q$. Secondly, elliptic curves E over $\mathbb{F}_q$ with $\#E(\mathbb{F}_q)=q$ are susceptible to the attack described by Semaev [80], Smart [82], and Satoh and Araki [77]. This attack efficiently maps the elliptic curve E into the additive group of $\mathbb{F}_q$. Thirdly, elliptic curves E over $\mathbb{F}_{2^m}$ with m composite may be susceptible to attacks based on the Weil descent - see the work of Galbraith and Smart [34], and Gaudry, Hess, and Smart [36]. These weak classes of curves are excluded in this document.

Additional information on the difficulty of the ECDLP can be found in ANSI X9.62 [3], ANSI X9.63 [4],

Blake, Seroussi, and Smart [12], and Koblitz, Menezes, and Vanstone [55]. A useful source of information on the state-of-the-art in practical attacks on the ECDLP is Certicom's ECC challenge [21].

The efficiency of cryptographic schemes based on ECC usually rests primarily on the efficiency of field operation computations and in particular point scalar multiplication computation. Efficient general techniques for computing field operations are well-known and are described, for example, in [51, 63, 67]. A variety of efficient general techniques for computing point scalar multiplication are known and exploit tricks like precomputation and switching to projective co-ordinates.

Both field operation computations and point scalar multiplication computation can be accelerated by choosing particular underlying fields and elliptic curves. Examples of fields amenable to particularly efficient implementation include $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ where p is a Mersenne prime - see, for example [6, 7, 8, 31]. Examples of elliptic curves amenable to particularly efficient implementation include Koblitz curves over $\mathbb{F}_{2^m}$ [53] which possess an efficiently computable endomorphism.

Additional information on the implementation of efficient finite field operations and point scalar multiplication can be found in Blake, Seroussi, and Smart [12], and Koblitz, Menezes, and Vanstone [55].

B.2 Commentary on Section 3 - Cryptographic Components

This section provides a commentary on Section 3 of the main body of this document.

B.2.1 Commentary on Elliptic Curve Domain Parameters

Elliptic curve domain parameters must be generated during the setup procedure of each of the schemes specified in this document.

The first step in this process is to determine the security level desired by the application in question. A number of criteria may affect this determination - factors may include, for example, the value of the information which the scheme will be used to protect, the length of time the parameters will be used for, and the security level of other schemes used in the application. The definitive paper of Blaze, Diffie, Rivest, Schneier, Shimomura, Thompson, and Wiener [16] provides guidance which may help implementers make this determination. In particular, the authors of [16] conclude that:

> To provide adequate protection against the most serious threats - well-funded commercial enterprises or government intelligence agencies - keys used to protect data today should be at least 75 bits long. To protect information adequately for the next 20 years in the face of expected advances in computing power, keys in newly-deployed systems should be at least 90 bits long.

Table 3 below attempts to provide additional information which may help determine the security level desired. It lists comparable key sizes for an 'ideal' symmetric scheme, an ECC-based scheme, and a scheme such as DSA or RSA based on the integer factorization problem or traditional discrete logarithm

problem. The figures in Table 3 are based on the running times of the best algorithms known in 2000 for attacking the cryptographic schemes.

Security level (in bits)	Symmetric scheme (key size in bits)	ECC-based scheme (size of n in bits)	DSA/RSA (modulus size in bits)
56	56	112	512
80	80	160	1024
112	112	224	2048
128	128	256	3072
192	192	384	7680
256	256	512	15360

Table 3: Comparable key sizes

Once the desired security level has been selected, there are a number of ways to generate elliptic curve domain parameters at a given strength. These include:

- Select an appropriate finite field. Then select an elliptic curve over the field at random, count the number of points on the curve using Schoof's algorithm [79], check whether the number of points is nearly prime, and repeat until appropriate parameters are found.
- Select an appropriate field, then select an appropriate curve order, and generate a curve over the field with this number of points using techniques based on 'complex multiplication' [59].
- Select an appropriate finite field. Then select an elliptic curve over the field from a special family of curves whose order is easily computable (such as the family of Koblitz curves), compute the number of points on this curve, and check whether the number of points is nearly prime.

The first method - known as selecting elliptic curve domain parameters at random - is the most conservative choice because it offers a probabilistic guarantee against future special purpose attacks of a similar nature to the Menezes-Okamoto-Vanstone and the Semaev-Smart-Satoh-Araki attacks described in Section B.1. However existing implementations of Schoof's algorithm are less efficient than implementations of the other parameter selection methods. The second method - known as selecting elliptic curve domain parameters using complex multiplication - generates parameters more efficiently than the first method. The third method - known as selecting elliptic curve domain parameters from a special family - also generates parameters more efficiently than the first method, and has the added attraction than some special families of curves (such as the family of Koblitz curves) enable tricks to speedup computations like point scalar multiplication. However despite their efficiency benefits, the second and third methods should be used with a good deal of caution because they produce parameters which may be susceptible to future special-purpose attacks.

An attractive refinement of the idea of selecting elliptic curve domain parameters at random is the idea of selecting elliptic curve domain parameters verifiably at random. This involves selecting parameters at random from a seed in such a way that the parameters cannot be pre-determined. It is appealing because the seed provides evidence that can be verified by anyone that no 'trapdoors' have been placed in the parameters. One method of selecting parameters verifiably at random is specified in ANSI X9.62 [3].

SEC 2 [83] - a companion document to this document - provides a list of precomputed elliptic curve domain parameters at a variety of commonly required security levels that implementers may use when implementing the schemes in this document. Use of these precomputed parameters is strongly recommended in order to foster interoperability.

Once elliptic curve domain parameters have been generated, either by the user themselves' or by a third party, it is desirable to receive some assurance that the parameters are valid - that is that the parameters possess the arithmetic properties expected from secure parameters. Parameter validation mitigates against inadvertent generation of insecure parameters caused by coding errors, and against deliberate attempts to trick users into using insecure parameters.

Additional information on the generation and validation of elliptic curve domain parameters can be found in ANSI X9.62 [3], ANSI X9.63 [4], and IEEE P1363 [40].

B.2.2 Commentary on Elliptic Curve Key Pairs

Elliptic curve key pairs must be generated during the operation of each of the schemes specified in this document. The key pair generation process requires a secure random or pseudorandom number generator. Design of secure random and pseudorandom number generators is notoriously difficult and implementers should therefore take care to pay attention to this aspect of their system design.

Once a key pair has been generated, it is often necessary to convey the public key in an authentic manner to other entities. One way of achieving this authentic distribution is to have the key certified by a trusted Certification Authority within a Public Key Infrastructure.

In many schemes it is desirable for an entity to receive assurance that an elliptic curve public key is valid or partially valid before they use the public key to, say, verify a signature. This process can help prevent small subgroup attacks and other attacks based on the use of an invalid public key.

A further discussion of the generation and validation of elliptic curve key pairs can be found in ANSI X9.63 [4].

B.2.3 Commentary on Elliptic Curve Diffie-Hellman Primitives

Both the elliptic curve Diffie-Hellman primitives here generate a field element from a secret key owned by one entity U and a public key owned by a second entity V in such a way that if both entities execute the primitive with corresponding keys as input, they will both compute the same field element.

The primary security requirement of both the primitives is that an attacker who sees only U and V's public keys should be unable to compute the shared field element. This requirement is equivalent to the

requirement that the elliptic curve Diffie-Hellman problem or ECDHP is hard. The ECDHP is stated as follows.

Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, and let $G \in E(\mathbb{F}_q)$ be a point on E on large prime order n. The ECDHP is, given E, G, and two scalar multiples $Q_1 = d_1G$ and $Q_2 = d_2G$ of G, to determine d_1d_2G.

The ECDHP is closely related to the ECDLP. It is clear that if the ECDLP is easy then so is the ECDHP. Boneh and Lipton [19] show that for practical purposes the converse is also true - they show that if the best algorithm for the ECDLP really is fully exponential, then the ECDLP and the ECDHP are equivalent.

Many schemes based on the Diffie-Hellman primitives actually rely on a stronger requirement that the shared field element is not just hard for an attacker to predict, but that the element actually looks random to the attacker. A discussion of this requirement, and its relationship to the ECDHP, can be found in [17, 20].

So far the discussion of the elliptic curve Diffie-Hellman primitives has been germane to both the 'standard' primitive and the cofactor primitive. The remainder of this section explains the difference between the two primitives.

A direct assault on the ECDHP is not the only way an attacker might assault schemes which use the Diffie-Hellman primitives. Many schemes which use the primitives are also susceptible to small subgroup attacks [48, 62] in which an adversary substitutes V's public key with a point of small order in an attempt to coerce U to calculate a predictable field element using one of the primitives. The consequences of these attacks can be severe - in a key agreement scheme, for example, the result can be compromise of a session key shared by U and V, or even compromise of U's static secret key.

Two defenses against this attack are recommended here: either validate V's public key and use the 'standard' Diffie-Hellman primitive (validating V's public key checks that V's public key has order n and hence prevents the attack), or partially validate V's public key and use the cofactor Diffie-Hellman primitive (using the cofactor Diffie-Hellman primitive with a point in a small subgroup will result in calculation of the point at infinity and hence rejection of the key).

Which of the defenses outlined above is appropriate in a given situation will depend on issues like whether or not interoperability with existing use of the 'standard' Diffie-Hellman primitive is desirable (the first defense interoperates and the second does not), and what the efficiency requirements of the system are (the second defense is usually more efficient than the first).

Additional information on the elliptic curve Diffie-Hellman primitives can be found in ANSI X9.63 [4].

B.2.4 Commentary on the Elliptic Curve MQV Primitive

The elliptic curve MQV primitive generates a field element from two key pairs owned by one entity U and two public keys owned by a second entity V in such a way that if both entities execute the primitive with corresponding keys as input, they will both compute the same field element.

Again the primary security requirement of the primtive is that an adversary who sees only U and V's public keys should be unable to compute the shared field element. This requirement is conjectured to be

equivalent to the requirement that the ECDHP is hard - see [60] for a discussion of this conjecture.

Additional information on the elliptic curve MQV primitive can be found in ANSI X9.63 [4].

B.3 Commentary on Section 4 - Signature Schemes

This section provides a commentary on Section 4 of the main body of this document.

B.3.1 Commentary on the Elliptic Curve Digital Signature Algorithm

The ECDSA is a signature scheme with appendix based on ECC. It is designed to be existentially unforgeable, even in the presence of an adversary capable of launching chosen-message attacks. ECDSA is an elliptic curve analog of the U.S. government's Digital Signature Algorithm or DSA [31]. It was first proposed in ANSI X9.62 [3].

The ECDSA was chosen for inclusion in this document because it is widely standardized in, for example, ANSI X9.62 [3], FIPS 186-2 [31] IEEE P1363 [40], and ISO 15946-2 [44]. Its widespread standardization, together with its close relationship to DSA, means that both specification details and implementation details have been carefully scrutinized. Standardization also leads to the provision of valuable tools like NIST's proposed ECDSA validation system which implementers will be able to use to check their code for errors.

The features of ECDSA outlined above were considered to outweigh at this time the features of other candidates like the Schnorr scheme [78], which was shown to be provably secure in the random oracle model based on the ECDLP in [75], or the Nyberg-Rueppel scheme [69, 70], which avoids the need for modular inversion during signature generation and verification and can offer slightly smaller signature sizes through its message recovery capability. Future consideration may be given to adding support for a signature scheme with features like provable security or added efficiency if significant commercial interest in these features is forthcoming.

There are a number of known cryptographic attacks on ECDSA. The specification of ECDSA in this document includes provision for preventing all of these attacks. Nonetheless implementers should be aware of the attacks and monitor future advances. The attacks illustrate that it is important that implementations of ECDSA perform all the security checks specified in the main body of this document. The following is a list of some of the known attacks:

- Attacks on the ECDLP. The security of ECDSA relies on the difficulty of the ECDLP on the elliptic curve domain parameters being used - otherwise an attacker may be able to recover U's secret key from U's public key and use this information to forge U's signature on any message.
- Attacks on key generation. Key generation is involved in both the key deployment procedure and the signing operation of ECDSA. Secure random or pseudorandom number generation is required during key generation to prevent, for example, U selecting a predictable secret key. Insecure random and pseudorandom number generators are perhaps the most common cause of cryptographic attacks on cryptographic systems.

- Attacks on the hash function. The hash function used by ECDSA during the signing operation and the verifying operation must possess a number of properties such as one-wayness and collision resistance. Otherwise if the hash function is not, say, collision resistant, an attacker may be able to find a collision (M_1, M_2) and forge U's signature on M_2 after persuading U to sign M_1.

- Attacks based on invalid domain parameters. The security of ECDSA relies on U using valid domain parameters because, for example, invalid domain parameters may be susceptible to the Pohlig-Hellman attack [73]. U should therefore receive assurance that the elliptic curve domain parameters used are valid. V may also desire to check that the elliptic curve domain parameters are valid to prevent attacks like those described in [15, 22] and to mitigate against the possibility of repudiation disputes in which U denies liability because U was using invalid domain parameters.

- Attacks based on invalid public keys. It may be desirable for V to check that U's public key is valid to prevent, for example, attacks like those described in [15, 22]. Another reason V may wish to check that U's public key is valid is to mitigate against the possibility of a repudiation dispute in which U denies liability because U was using an invalid public key.

- Vaudenay's attack. Vaudenay has shown in [86] that ECDSA is susceptible to attack if an attacker is able to persuade U to use elliptic curve domain parameters with base point order n chosen by the attacker and satisfying $\lceil \log_2 n \rceil \leq 8(hashlen)$. This attack can be prevented, for example, through exclusive use of elliptic curve domain parameters generated by U or by some trusted party, through use of verifiably random elliptic curve domain parameters or elliptic curve domain parameters from a small well-known family of parameters like parameters associated with Koblitz curves, or through use of parameters with $\lceil \log_2 n \rceil > 8(hashlen)$.

Of course a variety of non-cryptographic attacks on ECDSA are also possible, and implementers should take precautions to avoid, for example, 'implementation attacks' such as fault-based attacks [18], power-analysis attacks [56], and timing-analysis attacks [57].

The operation of ECDSA involves selection by implementers of a number of options. These choices will typically be made based on concerns like efficiency, interoperability, and on security issues like those outlined above. In particular, some of the choices involved are selection of elliptic curve domain parameters, selection of a hash function (SHA-1 is the only hash function currently supported), and selection of parameter and public key validation methods. Selection of elliptic curve domain parameters will likely involve consideration of issues like those discussed in Sections B.1 and B.2, and selection of parameter and public key validation methods will likely involve consideration of issues like those discussed above.

Additional information on ECDSA, including extensive security discussion, can be found in ANSI X9.62 [3] and IEEE P1363 [40]. Test vectors for ECDSA can be found in GEC 1 [37].

B.4 Commentary on Section 5 - Encryption Schemes

This section provides a commentary on Section 5 of the main body of this document.

B.4.1 Commentary on the Elliptic Curve Integrated Encryption Scheme

The ECIES is a public-key encryption scheme based on ECC. It is designed to be semantically secure in the presence of an adversary capable of launching chosen-plaintext and chosen-ciphertext attacks. It was proposed in [1, 11].

(Note that the specification of ECIES here differs slightly from the description in [1] where it is mandated that R is included in the input to the key derivation function. This does not affect the applicability of the security results in [1] to ECIES because elliptic curve domain parameters specify use of a prime order group generated by the base point G. Nevertheless implementations may of course choose to include R in the input to the key derivation function to achieve complete alignment with [1].)

The ECIES was chosen for inclusion in this document because it offers an attractive mix of provable security and efficiency. It was proven secure based on a variant of the Diffie-Hellman problem in [1]. It is as efficient as or more efficient than comparable schemes. The dominant calculations involved in encryption are two point scalar multiplications, and the dominant calculation involved in decryption is a single point scalar multiplication. ECIES is also being standardized in ANSI X9.63 [4] and IEEE P1363A [41].

The features of ECIES outlined above were considered to make it the most attractive ECC-based public-key encryption scheme for standardization. In particular, of the other possibilities, the elliptic curve analog of traditional ElGamal encryption [27] does not offer security against chosen-ciphertext attacks, while the elliptic curve analog of the Cramer-Shoup encryption scheme [23] offers similar security properties but is considerably less efficient.

There are a number of known attacks on ECIES. The specification of ECIES in this document includes provision for preventing all these attacks. Nonetheless implementers should be aware of the attacks and monitor future advances. The attacks illustrate that it is important that implementations of ECIES perform all the security checks specified in the main body of this document. The following is a list of some of the known attacks:

- Attacks on the ECDLP or ECDHP. The security of the ECIES relies on the difficulty of the ECDLP and ECDHP on the elliptic curve domain parameters used - otherwise an attacker who sees an encrypted message sent from U to V may be able to recover the shared secret value z from R and Q_V, and use this information to discover the message.

- Attacks on key generation. Key generation is involved in both the key deployment procedure and the encryption operation of ECIES. Secure random or pseudorandom number generation is required during key generation to prevent, for example, V selecting a predictable secret key. Insecure random and pseudorandom number generators are perhaps the most common cause of cryptographic attacks on cryptographic systems.

- Attacks on the symmetric encryption scheme. The symmetric encryption scheme used by the ECIES needs to possess only mild security properties to ensure the security of ECIES. (That is why the XOR encryption scheme may be used by ECIES.) Nonetheless severe compromise of the symmetric encryption scheme may result in leakage of information about encrypted messages.

- Attacks on the MAC scheme. As was the case for the symmetric encryption scheme, the MAC scheme used by ECIES needs to possess only mild security properties to ensure the security of ECIES. Nonetheless severe compromise of the MAC scheme may enable an attacker to launch a chosen-ciphertext attack on ECIES.

- Attacks on the key derivation function. The key derivation function used by ECIES must possess a number of properties to ensure the security of ECIES. If, for example, an attacker is able to predict some bits of the output of the key derivation function, or if portions of the output of the key derivation function are correlated in some way, an attacker may be able to learn some information about encrypted messages. These concerns provide some motivation for the use of the TDES symmetric encryption scheme rather than the XOR symmetric encryption scheme when using ECIES to convey long messages since this choice minimizes the amount of output the key derivation function is asked to produce.

- Attacks based on the use of invalid domain parameters. The security of ECIES relies on V using valid domain parameters because, for example, invalid domain parameters may be susceptible to the Pohlig-Hellman attack [73]. V should therefore receive assurance that the elliptic curve domain parameters used are valid.

- Attacks based on the use of invalid public keys. When the ECIES is used with the standard elliptic curve Diffie-Hellman primitive, V should check that the public-key R is valid to prevent Lim-Lee style small subgroup attacks [48, 62] which result in providing an attacker with the ability to learn some bits of V's secret key. Similarly when ECIES is used with the cofactor Diffie-Hellman primitive, V should check that the public-key R is partially valid to prevent the possibility of similar attacks. (The cofactor Diffie-Hellman primitive is designed specifically to enable efficient prevention of small subgroup attacks.)

Of course a variety of non-cryptographic attacks on ECIES are also possible, and implementers should take precautions to avoid, for example, 'implementation attacks' such as fault-based attacks [18], power-analysis attacks [56], and timing-analysis attacks [57].

The operation of ECIES involves selection by implementers of a number of options. These choices will typically be made based on concerns like efficiency, interoperability, and on security issues like those outlined above. In particular, some of the choices involved are selection of elliptic curve domain parameters, selection of a key derivation function, selection of a symmetric encryption scheme and MAC scheme, selection of the standard or cofactor Diffie-Hellman primitive, selection of parameter and public key validation methods, and selection of appropriate data to include in $SharedInfo_1$ and $SharedInfo_2$. Selection of elliptic curve domain parameters will likely involve consideration of issues like those discussed in Sections B.1 and B.2. Selection of a symmetric encryption scheme will likely be influenced by the length of messages which are going to be encrypted and the amount of memory available. (When the TDES encryption scheme is used, messages can be passed into the encryption operation a piece at a time, whereas when the XOR encryption scheme is used to encrypt variable length messages, the length of the message must be known before MAC computation can begin.) Selection of the HMAC–SHA-1–160 scheme or the HMAC–SHA-1–80 scheme will likely be influenced by individual security preferences

and bandwidth requirements. Selection of the standard or cofactor Diffie-Hellman primitive will likely involve consideration of security concerns like small subgroup attacks and the efficiency requirements of the application. Selection of parameter and public key validation methods will likely involve consideration of security issues like those discussed above. Selection of appropriate information to include in $SharedInfo_1$ and $SharedInfo_2$ will likely depend on the particular application, but common things to include are, for example, the public key R for alignment with the description of ECIES in [1], or a counter value to mitigate against replay of ciphertexts.

Additional information on ECIES, including extensive security discussion, can be found in ANSI X9.63 [4], and the paper of Abdalla, Bellare, and Rogaway [1]. Test vectors for ECIES can be found in GEC 1 [37].

B.5 Commentary on Section 6 - Key Agreement Schemes

This section provides a commentary on Section 6 of the main body of this document.

B.5.1 Commentary on the Elliptic Curve Diffie-Hellman Scheme

The elliptic curve Diffie-Hellman scheme is a key agreement scheme based on ECC. It is designed to provide a variety of security goals depending on its application - goals it can provide include unilateral implicit key authentication, mutual implicit key authentication, known-key security, and forward secrecy. It is the elliptic curve analog of the Diffie-Hellman scheme [25]. It was first proposed in [25, 52, 68].

The elliptic curve Diffie-Hellman scheme was chosen for inclusion in this document because it is well-known, well-scrutinized, widely-standardized, and versatile. It is standardized in ANSI X9.63 [4], IEEE P1363 [40], and ISO 15946-3 [45]. Examples of the application of the elliptic curve Diffie-Hellman scheme to achieve a variety of security goals can be found in [10, 14, 26]. ECIES is also an example of an application of the elliptic curve Diffie-Hellman scheme.

There are a number of known attacks on the elliptic curve Diffie-Hellman scheme. The specification of the elliptic curve Diffie-Hellman scheme in this document includes provision for preventing all these attacks. Nonetheless implementers should be aware of the attacks and monitor future advances. The attacks illustrate that it is important that implementations of the elliptic curve Diffie-Hellman scheme perform all the security checks specified in the main body of this document. The following is a list of some of the known attacks:

- Attacks on the ECDLP or ECDHP. The security of the elliptic curve Diffie-Hellman scheme relies on the difficulty of the ECDLP and ECDHP on the elliptic curve domain parameters used - otherwise an attacker who sees an U to V using the scheme may be able to recover the shared secret value z from Q_U and Q_V, and use this information to discover the keying data they agreed.

- Attacks on key generation. Key generation is involved in the key deployment procedure of the elliptic curve Diffie-Hellman scheme. Secure random or pseudorandom number generation is required

during key generation to prevent, for example, V selecting a predictable secret key. Insecure random and pseudorandom number generators are perhaps the most common cause of cryptographic attacks on cryptographic systems.

- Man-in-the-middle attacks. If the elliptic curve Diffie-Hellman scheme is not applied with care, it may be possible for an adversary to attack the scheme by modifying Q_U or Q_V when they are exchanged, and as a result prevent the scheme from achieving goals like implicit key authentication or known-key security. Numerous defenses are commonly employed to prevent such active attacks - including exchanging Q_U and Q_V in signed messages, or certifying Q_U and Q_V.

- Attacks on the key derivation function. The key derivation function used by the elliptic curve Diffie-Hellman scheme must possess a number of properties to ensure the security of the scheme. If, for example, an attacker is able to predict some bits of the output of the key derivation function, or if portions of the output of the key derivation function are correlated in some way, an attacker may be able to learn some information about the agreed keying data.

- Attacks based on the use of invalid domain parameters. The security of the elliptic curve Diffie-Hellman scheme relies on U and V using valid domain parameters because, for example, invalid domain parameters may be susceptible to the Pohlig-Hellman attack [73]. U and V should therefore receive assurance that the elliptic curve domain parameters used are valid.

- Attacks based on the use of invalid public keys. Because the elliptic curve Diffie-Hellman scheme by its nature requires each entity to combine its private key with another entity's public key, the scheme is particularly susceptible to attacks based on the use of invalid public keys. The best-known examples of such attacks are small subgroup attacks [48, 62], which can result in, for example, an attacker coercing U and V into sharing predictable keying data, or an attacker learning some bits of U's secret key. For this reason, when using the elliptic curve Diffie-Hellman scheme with the standard Diffie-Hellman primitive, U should receive an assurance that V's public key is valid and vice versa, and when using the scheme with the cofactor Diffie-Hellman primitive, U should receive an assurance that V's public key is partially valid and vice versa.

Of course a variety of non-cryptographic attacks on the elliptic curve Diffie-Hellman scheme are also possible, and implementers should take precautions to avoid, for example, 'implementation attacks' such as fault-based attacks [18], power-analysis attacks [56], and timing-analysis attacks [57].

The operation of the elliptic curve Diffie-Hellman scheme involves selection by implementers of a number of options. These choices will typically be made based on concerns like efficiency, interoperability, and on security issues like those outlined above. In particular, some of the choices involved are selection of elliptic curve domain parameters, selection of a key derivation function, selection of the standard or cofactor Diffie-Hellman primitive, selection of parameter and public key validation methods, and selection of $SharedInfo$, as well as selection of an appropriate application of the scheme to meet the security requirements of the system. Selection of elliptic curve domain parameters will likely involve consideration of issues like those discussed in Section B.1 and B.2. Selection of the standard or cofactor Diffie-Hellman primitive will likely involve consideration of security concerns like small subgroup attacks and the efficiency requirements of the system. Selection of parameter and public key validation methods will likely

involve consideration of security issues like those discussed above. Selection of appropriate information to include in $SharedInfo$ will likely depend on the particular application, but common things to include are, for example, the identities of U and V, the public keys Q_U and Q_V, counter values, and an indication of the symmetric scheme for which the agreed keying data will be used. If a number of fields are included in $SharedInfo$, it is sensible to check that the encoding of the fields is unique. Selection of an appropriate application of the scheme will likely depend on issues like what the agreed key will be used for and whether U and V are both on-line. ANSI X9.63 contains guidance to help implementers make this selection.

Additional information on the elliptic curve Diffie-Hellman scheme, including extensive security discussion, can be found in ANSI X9.63 [4], and IEEE P1363 [40]. Test vectors for the elliptic curve Diffie-Hellman scheme can be found in GEC 1 [37].

B.5.2 Commentary on the Elliptic Curve MQV Scheme

Like the elliptic curve Diffie-Hellman scheme, the elliptic curve MQV scheme is a key agreement scheme based on ECC. It is designed to provide a variety of security goals depending on its application - goals it can provide include mutual implicit key authentication, known-key security, and forward secrecy. It was first proposed in [60, 66].

The elliptic curve MQV scheme was chosen for inclusion in this document because it is a particularly efficient method for achieving mutual implicit key authentication. The dominant calculations involved in the key agreement operation are 1.5 point scalar multiplies. The elliptic curve MQV scheme is also standardized in ANSI X9.63 [4], and IEEE P1363 [40].

There are a number of known attacks on the elliptic curve MQV scheme. The specification of the elliptic curve MQV scheme in this document includes provision for preventing all these attacks. Nonetheless implementers should be aware of the attacks and monitor future advances. The attacks illustrate that it is important that implementations of the elliptic curve MQV scheme perform all the security checks specified in the main body of this document. The following is a list of some of the known attacks:

- Attacks on the ECDLP or ECDHP. The security of the elliptic curve MQV scheme relies on the difficulty of the ECDLP and ECDHP on the elliptic curve domain parameters used - otherwise an attacker who sees an U to V using the scheme may be able to recover the shared secret value z from $Q_{1,U}$, $Q_{2,U}$, $Q_{1,V}$, and $Q_{2,V}$, and use this information to discover the keying data they agreed.
- Attacks on key generation. Key generation is involved in the key deployment procedure of the elliptic curve MQV scheme. Secure random or pseudorandom number generation is required during key generation to prevent, for example, V selecting a predictable secret key. Insecure random and pseudorandom number generators are perhaps the most common cause of cryptographic attacks on cryptographic systems.
- Attacks on the key derivation function. The key derivation function used by the elliptic curve MQV scheme must possess a number of properties to ensure the security of the scheme. If, for example, an attacker is able to predict some bits of the output of the key derivation function, or if portions

of the output of the key derivation function are correlated in some way, an attacker may be able to learn some information about the agreed keying data.

- Attacks based on the use of invalid domain parameters. The security of the elliptic curve MQV scheme relies on U and V using valid domain parameters because, for example, invalid domain parameters may be susceptible to the Pohlig-Hellman attack [73]. U and V should therefore receive assurance that the elliptic curve domain parameters used are valid.

- Unknown key-share attacks. Kaliski [50] has observed that the elliptic curve MQV scheme may be susceptible to unknown key-share attacks if it is not applied with care. These attacks may be damaging when the scheme is used to provide symmetric keys in order to both encrypt and authenticate data. The attacks can be prevented by including data like U and V's identities in $SharedInfo$, or by performing appropriate key confirmation subsequent to key agreement.

Of course a variety of non-cryptographic attacks on the elliptic curve MQV scheme are also possible, and implementers should take precautions to avoid, for example, 'implementation attacks' such as fault-based attacks [18], power-analysis attacks [56], and timing-analysis attacks [57].

The operation of the elliptic curve MQV scheme involves selection by implementers of a number of options. These choices will typically be made based on concerns like efficiency, interoperability, and on security issues like those outlined above. In particular, some of the choices involved are selection of elliptic curve domain parameters, selection of a key derivation function, selection of parameter and public key validation methods, and selection of $SharedInfo$, as well as selection of an appropriate application of the scheme to meet the security requirements of the system. Selection of elliptic curve domain parameters will likely involve consideration of issues like those discussed in Section B.1 and B.2. Selection of parameter and public key validation methods will likely involve consideration of security issues like those discussed above. Selection of appropriate information to include in $SharedInfo$ will likely depend on the particular application, but common things to include are, for example, the identities of U and V, the public keys $Q_{1,U}$, $Q_{2,U}$, $Q_{1,V}$, and $Q_{2,V}$, counter values, and an indication of the symmetric scheme for which the agreed keying data will be used. If a number of fields are included in $SharedInfo$, it is sensible to check that the encoding of the fields is unique. Selection of an appropriate application of the scheme will likely depend on issues like what the agreed key will be used for and whether U and V are both on-line. ANSI X9.63 contains guidance to help implementers make this selection.

Additional information on the elliptic curve MQV scheme, including extensive security discussion, can be found in ANSI X9.63 [4], in IEEE P1363 [40], and in the paper of Law, Menezes, Qu, Solinas, and Vanstone [60]. Test vectors for the elliptic curve MQV scheme can be found in GEC 1 [37].

B.6 Alignment with Other Standards

The cryptographic schemes in this document have been selected to conform with as many other standards efforts on ECC as possible.

Core standards efforts on ECC include ANSI X9.62 [3], ANSI X9.63 [4], FIPS 186-2 [31], IEEE P1363 [40], IEEE P1363A [41], IPSec [38, 72, 28], ISO 14888-3 [42], and ISO 15946 [43, 44, 45].

Table 4 shows which of the schemes specified in this document are included in these efforts. More details are given below.

Standard	Schemes included
ANSI X9.62	ECDSA
ANSI X9.63	ECIES, ECDH, ECMQV
FIPS 186-2	ECDSA
IEEE P1363	ECDSA, ECDH, ECMQV
IEEE P1363A	ECIES
IPSec	ECDSA, ECDH
ISO 14888-3	ECDSA
ISO 15946	ECDSA, ECDH, ECMQV

Table 4: Alignment with other core ECC standards

The ANSI X9.62 standard specifies ECDSA for use by the financial services industry. It requires ECDSA to be used with the hash function SHA-1, and with elliptic curve domain parameters with $n > 2^{160}$ to meet the stringent security requirements of the banking industry. Subject to these constraints and other procedural constraints like the use of an ANSI-approved pseudorandom number generator, the specification of ECDSA in this document should comply with ANSI X9.62.

The draft ANSI X9.63 standard specifies key agreement and key transport schemes based on ECC for use by the financial services industry. In particular it specifies various schemes built from ECIES, ECDH, and ECMQV. Like ANSI X9.62, it requires various constraints like restriction to the hash function SHA-1, use of elliptic curve domain parameters with $n > 2^{160}$, and use of an ANSI-approved pseudorandom number generator. Subject to these constraints, the specifications of ECDH and ECMQV in this document should be compatible with ANSI X9.63. The specification of ECIES in this document should be similarly compatible with ANSI X9.63 when used with the XOR symmetric encryption scheme. (ANSI X9.63 is concerned specifically with key transport of short keys and hence support for a TDES symmetric encryption option is not so desirable there as it is here where longer messages may be encrypted.)

The FIPS 186-2 standard specifies signature schemes for U.S. government use. Three signature schemes are allowed, including ECDSA as specified in ANSI X9.62. Thus subject to the constraints described above in the context of ANSI X9.62, the specification of ECDSA in this document should comply with FIPS 186-2.

The IEEE P1363 standard has a wide scope encompassing schemes based on the integer factorization problem, the traditional discrete logarithm problem, and the ECDLP. The techniques based on ECC specified in IEEE P1363 include general descriptions of ECDSA (known in IEEE P1363 as EC-SSA), ECDH (known as ECKAS-DH1), and ECMQV (known as ECKAS-MQV). The specifications of ECDSA, ECDH, and ECMQV in this document should comply with IEEE P1363.

The draft IEEE P1363A standard is a proposed addendum to IEEE P1363. IEEE P1363A includes additional techniques not originally considered for IEEE P1363 due to time constraints. ECIES is included in IEEE P1363A [41]. The specification of ECIES in this document should comply with IEEE P1363A.

The IPSec standard [38] includes support for a variant of ECDH. The particular variant of ECDH differs from ECDH as specified in this document in as much as it uses different octet string point representations. In addition the default elliptic curve domain parameters in IPSec are parameters over $\mathbb{F}_{2^m}$ with m composite and they do not use prime order base points. Aside from these technicalities, the specification of ECDH in this document should be broadly compliant with IPSec. Furthermore, draft IPSec standards [28, 72] have been proposed which extend IPSec to include ECDSA and ECDH as specified in this document.

The ISO 14888-3 standard specifies very general signature mechanisms. The specification of ECDSA in this document should comply with ISO 14888-3.

The draft ISO 15946 standards specify cryptographic techniques based on ECC. ISO 15946-2 specifies a variety of signature schemes including ECDSA. The specification of ECDSA in this document should comply with ISO 15946-2. ISO 15946-3 specifies a variety of key establishment schemes including some based on ECDH and ECMQV. The specifications of ECDH and ECMQV in this document should be compatible with ISO 15946-3.

A variety of standards build on the core standards above. In particular: The FSTC FSML standard [33] allows ECDSA as specified in ANSI X9.62 to be used to sign electronic checks and other electronic financial documents. The PKIX standard [9] specifies how to use ECDSA as specified in ANSI X9.62 within the PKIX certification framework. The S/MIME standard [13] specifies how to use ECDSA, ECDH, and ECMQV as specified in ANSI X9.62 and ANSI X9.63 to secure email. The SSL/TLS ECC standard [24] specifies how to use ECDSA, ECDH, and ECMQV as specified in ANSI X9.62, ANSI X9.63, and IEEE P1363 in SSL/TLS. The WAP WTLS standard [87] specifies a protocol similar in spirit to SSL/TLS but tailored to the needs of the wireless telecommunications industry. It allows use of ECDSA and ECDH as specified in IEEE P1363. The WAP WMLScript Crypto API standard [88] specifies application layer cryptographic functions for use by wireless devices. It allows the use of ECDSA as specified in IEEE P1363. From the discussion above on the alignment of this document with other core standards, it can be seen that the schemes in this document should be appropriate for use by implementers of the FSTC FSML standard, the IPSec standards, the PKIX standards, the S/MIME standards, the SSL/TLS ECC standard, and the WAP standards.

C ASN.1 for Elliptic Curve Cryptography

Several different types of information may need to be conveyed during the operation of the schemes specified in this document. This section suggests ASN.1 syntax which may be used to convey parts of this information when suitably encoded via, for example, DER [47]. Section C.1 recommends syntax to describe finite fields. Section C.2 recommends syntax to describe elliptic curve domain parameters. Section C.3 recommends syntax to describe elliptic curve public keys. Section C.4 recommends syntax to describe elliptic curve private keys. Section C.5 recommends syntax to describe signatures. Section C.6 recommends syntax to encode information for processing by key derivation functions. Section C.7 contains the ASN.1 module that holds the above.

Syntax for other aspects of elliptic curve cryptography, such as object identifiers for specific schemes, may be added in future versions of this document. The syntax provided here follows the syntax used in ANSI X9.62 [3], ANSI X9.63 [4], and PKIX [9].

C.1 Syntax for Finite Fields

This section provides the recommended ASN.1 syntax to identify finite fields and field elements. The identity of a finite field and a specific field element therein may need to be specified, for example, as part of some elliptic curve domain parameters. The syntax follows ANSI X9.62 [3, Section 7.1].

The finite fields of interest in this document are prime fields and characteristic-two fields. A finite field is identified by a value of type `FieldID`:

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

The governor `FIELD-ID` of the parameter `FIELD-ID:IOSet` is defined as the following open type that is defined in ITU-T X.681 [46, Annex A].

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

(and hence the dummy argument `IOSet` must be of said type). The term “open type” means that a ‘hole’ is left in both components that are filled at the time of usage. The component relation constraint {`IOSet`}{`@fieldType`} binds the argument `IOSet` to the identifier `fieldType`.

Only two field types are permitted, namely, prime fields and characteristic-two fields.

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

A prime field is specified by the identifier `prime-field` of type `Prime-p` (below) comprising an integer which is the size of the field.

```
prime-field  OBJECT IDENTIFIER ::= { id-fieldType 1 }
Prime-p ::= INTEGER    -- Field of size p.
```

The object identifier `id-fieldType` (above) is the root of a tree containing object identifiers of each field type. It is defined as the following arc from ANSI.

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

where

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

A characteristic-two finite field is specified by the identifier `characteristic-two-field` of type `Characteristic-two` (below) comprising the size of the field, the type of basis used to express elements of the field, and the polynomial used to generate the field (in the case of a polynomial basis).

```
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})
}
```

The type `CHARACTERISTIC-TWO` (above) is defined by the following.

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

The basis types of interest are normal bases (that are not used here), trinomial bases, or pentanomial bases. (See Section 2.1.2 for further information.)

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

Normal bases are specified by the object identifier `gnBasis` (below) with `NULL` parameters. Trinomial bases are specified by the object identifier `tpBasis` (below) with a parameter `Trinomial` that specifies the degree of the middle term in the defining trinomial. Pentanomial bases are specified by the object identifier `ppBasis` (below) with a parameter `Pentanomial` that specifies the degrees of the three middle terms in the defining pentanomial.

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

The identifier `id-characteristic-two-basis` (above) is defined as the following.

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

The degrees of the polynomials that define the finite fields are specified by the following.

```
Trinomial ::= INTEGER
Pentanomial ::= SEQUENCE {
   k1 INTEGER,  -- k1 > 0
   k2 INTEGER,  -- k2 > k1
   k3 INTEGER   -- k3 > k2
}
```

Finally, a specific field element is represented by the following type

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

whose value is the octet string obtained from the conversion routines given in Section 2.3.5.

C.2 Syntax for Elliptic Curve Domain Parameters

Elliptic curve domain parameters may need to be specified, for example, during the setup operation of a cryptographic scheme based on elliptic curve cryptography. There are a number of ways of specifying elliptic curve domain parameters. Here the following syntax is recommended (following [9]) for use in X.509 certificates and elsewhere.

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

The choice of three parameter representation methods (above) allows detailed specification of all required values using either:

- The choice `ecParameters` which explicitly identifies all the parameters, or

- The choice `namedCurve` which implicitly identifies the parameters by reference to a well-known set of parameters, or
- The choice `implicitCA` which indicates that the parameters are inherited from elsewhere.

The three choices described above are completely specified in the remainder of this section.

The choice `ecParameters` explicitly identifies the elliptic curve domain parameters, if need be, via the following syntax.

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

The components of type `ECParameters` have the following meanings.

- The component `version` is the version number of the ASN.1 type with a value of `1`. The notation above creates an `INTEGER` named `ecpVer1` and assigns to it the value one. It is used to constrain `version` to a single value.
- The component `fieldID` identifies the finite field over which the elliptic curve is defined and was discussed in Section C.1.
- The component `curve` of type `Curve` (defined below) specifies the elliptic curve.
- The component `base` of type `ECPoint` (defined below) specifies the base point on the elliptic curve `curve`.
- The component `order` is the order of the base point `base`.
- The component `cofactor` is the order of the curve divided by the order of the base point. Inclusion of the cofactor is optional – however, it is strongly recommended that that the cofactor be included in order to facilitate interoperability between implementations.

The type `Curve` is defined as follows, by specifying the coefficients of the defining equation of the elliptic curve and an optional seed. (If the curve was generated verifiably at random using a seed value with SHA-1 as specified in ANSI X9.62 [3] then said seed may be included as the `seed` component so as to allow a recipient to verify that the curve was indeed so generated using said seed.)

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

An elliptic curve point itself is represented by the following type

```
ECPoint ::= OCTET STRING
```

whose value is the octet string obtained from the conversion routines given in Section 2.3.3.

The choice `namedCurve` implicitly identifies the elliptic curve domain parameters by reference to a well-known set of parameters assigned an object identifier.

The class `CURVES`, defined as follows, is used to specify the `namedCurve` choice.

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

As an example of the above syntax, the curve `sect163k1`, defined in SEC 2 [83], is denoted by the syntax `ID sect163k1`.

The beauty of the above syntax is that valid values for the choice `namedCurve` can be constrained. The following syntax, included here for completeness, may be extended by other standards and implementations to constrain the list of supported named curves. One such extension may be found in SEC 2 [83].

```
SECGCurveNames CURVES ::= {
    ... -- named curves
}
```

The choice `implicitlyCA` indicates that elliptic curve domain parameters are inherited from elsewhere. This choice must be used with care because it is important that elliptic curve key pairs are bound to the correct parameters so that an attacker is not able to coerce the use of the key pair with a different set of parameters.

Situations where use of the `implicitlyCA` choice may be appropriate include applications which use fixed parameters and applications where user parameters are inherited from Certificate Authority parameters.

C.3 Syntax for Elliptic Curve Public Keys

Elliptic curve public keys may need to be specified, for example, during the key deployment phase of a cryptographic scheme based on elliptic curve cryptography. An elliptic curve public key is a point on an elliptic curve and may be represented in a variety of ways using ASN.1 syntax. Here the following syntax is recommended (following [9]) for use in X.509 certificates and elsewhere, where public keys are represented by the ASN.1 type `SubjectPublicKeyInfo`.

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

The component `algorithm` specifies the type of public key and associated parameters employed and the component `subjectPublicKey` specifies the actual value of said public key.

The parameter type `AlgorithmIdentifier` above tightly binds together a set of algorithm object identifiers and their associated parameters types. The type `AlgorithmIdentifier` is defined as follows.

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

The governing type `ALGORITHM` (above) is defined to be the following object information object.

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

When `SubjectPublicKeyInfo` is used to specify an elliptic curve public key, the sole parameter in the reference of type `AlgorithmIdentifier` refers to the object `ecPublicKeyType`.

```
ECPKAlgorithms ALGORITHM ::= {
   ecPublicKeyType,
   ...
}
ecPublicKeyType ALGORITHM ::= {
   OID id-ecPublicKey PARMS Parameters {{SECGCurveNames}}
}
```

The object identifier `id-ecPublicKey` designates an elliptic curve public key. It is defined by the following (after ANSI X9.62 [3, Section 6.1]) to be used whenever an object identifier for an elliptic curve public key is needed. (Note that this syntax applies to all elliptic curve public keys regardless of their designated use.)

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

where

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

The component `Parameters` was defined in Section C.2 and may contain the elliptic curve domain parameters associated with the public key in question. (Thus the component `algorithm` indicates that `SubjectPublicKeyInfo` not only specifies the elliptic curve public key but also the elliptic curve domain parameters associated with said public key.)

Finally, `SubjectPublicKeyInfo` specifies the public key itself when `algorithm` indicates that the public key is an elliptic curve public key.

The elliptic curve public key (a value of type `ECPoint` that is an `OCTET STRING`) is mapped to a `subjectPublicKey` (a value encoded as type `BIT STRING`) as follows: The most significant bit of the value of the `OCTET STRING` becomes the most significant bit of the value of the `BIT STRING` and so on with consecutive bits until the least significant bit of the `OCTET STRING` becomes the least significant bit of the `BIT STRING`. Note that when DER-encoding is being used, this means that the raw public key value without prepended DER-encoding tag and length octets is placed in the `subject-PublicKey` field.

C.4 Syntax for Elliptic Curve Private Keys

An elliptic curve private key may need to be conveyed, for example, during the key deployment operation of a cryptographic scheme in which a Certification Authority generates and distributes the private keys. An elliptic curve private key is an unsigned integer. The following ASN.1 syntax may be used.

```
ECPrivateKey{CURVES:IOSet} ::= SEQUENCE {
    version INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
    privateKey OCTET STRING,
    parameters [0] Parameters{{IOSet}} OPTIONAL,
    publicKey  [1] BIT STRING OPTIONAL
}
```

where

- The component `version` specifies the version number of the elliptic curve private key structure. The syntax above creates the element `ecPrivkeyVer1` of type `INTEGER` whose value is `1`.

- The component `privateKey` is the private key defined to be the octet string of length $\lceil \log_2 n/8 \rceil$ (where n is the order of the curve) obtained from the unsigned integer via the encoding of Section 2.3.7.
- The optional component `parameters` specifies the elliptic curve domain parameters associated to the private key. The type `Parameters` was discussed in Section C.2. If the parameters are known by other means then this component may be `NULL` or omitted.
- The optional component `publicKey` contains the elliptic curve public key associated with the private key in question. Public keys were discussed in Section C.3. It may be useful to send the public key along with the private key, especially in a scheme such as MQV that involves calculations with the public key.

Note that omission of the `parameters` field of `ECPrivateKey` should be performed with care because it is important that elliptic curve key pairs are bound to the correct parameters so that an attacker is not able to coerce the use of the key pair with a different set of parameters.

The syntax for `ECPrivateKey` may be used, for example, to convey elliptic curve private keys using the syntax for `PrivateKeyInfo` as defined in PKCS #8 [74]. In this instance, the value of the component `privateKeyAlgorithm` within `PrivateKeyInfo` shall be `id-ecPublicKey` as discussed in section C.3 above. Also in this instance, it is recommended that the elliptic curve domain parameters are placed in the `privateKeyAlgorithm` field and the `parameters` field of `ecPrivateKey` is omitted.

C.5 Syntax for Signatures

Signatures may need to be conveyed from one party to another whenever ECDSA is used to sign a message. The following syntax is recommended to represent actual signatures for use within X.509 certificates, CRLs (following [9]), and elsewhere. The signature is conveyed using the parameterized type `SIGNED`. It comprises the specification of an algorithm of type `AlgorithmIdentifier` together with the actual signature

When the signature is generated using ECDSA with SHA-1, the algorithm component shall contain the object identifier `ecdsa-with-SHA1` (defined below) and the parameters component shall contain `NULL`. (The elliptic curve domain parameters must then be obtained from some other source, for example, from the signer's certificate.) In such cases, the `ALGORITHM` `ecdsa-SHA1`, defined below, shall be used.

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

The actual value of an ECDSA signature computed using SHA-1, that is, a signature identified by `ecdsa-with-SHA1`, is encoded as follows.

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

X.509 certificates and CRLs represent a signature as a bit string; in such cases, the entire encoding of a value of `ECDSA-Sig-Value` is the value of said bit string. Note that when DER-encoding is being used, this means that the entire ECDSA signature including DER-encoding tag and length octets is placed in the `signatureValue` field.

C.6 Syntax for Key Derivation Functions

This section provides ASN.1 syntax that may be used to encode input to the key derivation functions specified in Section 3.6. Note that the use of ASN.1 syntax for this purpose is optional - however the use of ASN.1 syntax in this scenario can help to ensure that the encoding of information fields is unambiguous.

The input to the key derivation functions includes an octet string *SharedInfo*. *SharedInfo* may contain an encoding of `ASN1SharedInfo` as defined below.

```
ASN1SharedInfo ::= SEQUENCE {
   keyInfo          AlgorithmIdentifier,
   entityUInfo  [0] OCTET STRING OPTIONAL,
   entityVInfo  [1] OCTET STRING OPTIONAL,
   suppPubInfo  [2] OCTET STRING OPTIONAL,
   suppPrivInfo [3] OCTET STRING OPTIONAL
}
```

The components of type `ASN1SharedInfo` have the following meanings:

- `keyInfo` specifies the symmetric algorithm for which the derived key is to be used.
- `entityUInfo`, if present, specifies additional information about the scheme's initiator such as the entity's X.501 distinguished name, the entity's public key, etc.
- `entityVInfo`, if present, specifies additional information about the scheme's responder such as the entity's X.501 distinguished name, the entity's public key, etc.
- `suppPubInfo`, if present, specifies additional public information known to both entities involved in the operation of the scheme.
- `suppPrivInfo`, if present, specifies additional private information known to both entities involved in the operation of the scheme.

An example of the use of `ASN1SharedInfo` can be found in [13].

C.7 ASN.1 Module

The following comprises the ASN.1 module for those items not defined in ANSI X9.62 [3] or ANSI X9.63 [4].

```
CERTICOM {
   iso(1) identified-organization(3) certicom(132) module(1) ver(1)
}
DEFINITIONS EXPLICIT TAGS ::= BEGIN
    --
    -- EXPORTS All;
    --

    --
    -- Import the required type to define private keys.
    --
    IMPORTS Parameters FROM ANSI-X9-62;

    certicom-arc OBJECT IDENTIFIER ::= {
        iso(1) identified-organization(3) certicom(132)
    }
    --
    -- Private-key syntax using Parameters from ANSI X9.62.
    --
    ECPrivateKey ::= SEQUENCE {
        version INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
        privateKey OCTET STRING,
        parameters [0] Parameters OPTIONAL,
        publicKey  [1] BIT STRING OPTIONAL
    }
END
```

D References

This section lists the references cited in this document.

[1] M. Abdalla, M. Bellare, and P. Rogaway. DHAES: An encryption scheme based on the Diffie-Hellman problem. 1998. Full version of [11]. Available from: `http://www-cse.ucsd.edu/users/mihir/`

[2] ANSI X9.52-1998: *Triple Data Encryption Algorithm Modes of Operation*. American Bankers Association, 1998.

[3] ANSI X9.62-1998: *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)*. American Bankers Association, 1999.

[4] ANSI X9.63-199x: *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*. October, 1999. Working Draft.

[5] ANSI X9.71-199x: *Keyed Hash Message Authentication Code*. March, 1998. Working Draft.

[6] G. Agnew, T. Beth, R. Mullin, and S. Vanstone. Arithmetic operations in $GF(2^m)$. *Journal of Cryptology*, 6, pages 3–13, 1993.

[7] G. Agnew, R. Mullin, and S. Vanstone. An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$. *IEEE Journal on Selected Areas in Communications*, 11, pages 804–813, 1993.

[8] G. Agnew, R. Mullin, I. Onyszchuk, and S. Vanstone. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3, pages 63–79, 1991.

[9] L. Bassham, R. Housley, and W. Polk. Representation of Public Keys and Digital Signatures in Internet X.509 Public Key Infrastructure Certificates. July, 2000. Internet Draft. Available from: `http://www.ietf.org/`

[10] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. *Proceedings of the 30th Annual Symposium on the Theory of Computing*. 1998.

[11] M. Bellare and P. Rogaway. Minimizing the use of random oracles in authenticated encryption schemes. In *Proceedings of PKS'97*, 1997.

[12] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[13] S. Blake-Wilson and D. Brown. Use of ECC algorithms in CMS. September, 2000. Internet Draft. Available from: `http://www.ietf.org/`

[14] S. Blake-Wilson, D. Johnson, and A.J. Menezes. Key agreement protocols and their security analysis. In *Proceedings of the sixth IMA International Conference on Cryptography and Coding*, pages 30–45, 1997.

[15] S. Blake-Wilson and A.J. Menezes. Unknown key-share attacks on the Station-to-Station (STS) protocol. In *Public Key Cryptography: PKC '99*, pages 154–170, 1999.

[16] M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener. *Minimal key lengths for symmetric ciphers to provide adequate commercial security.* January, 1996.

[17] D. Boneh. The decision Diffie-Hellman problem. In *Algorithmic Number Theory III*, pages 48–63, 1998.

[18] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology – EUROCRYPT '97*, pages 37–51, 1997.

[19] D. Boneh and R.J. Lipton. Algorithms for black-box fields and their application to cryptography. In *Advances in Cryptology: Crypto '96*, pages 283–297, 1996.

[20] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Advances in Cryptology: Crypto '96*, pages 129–142, 1996.

[21] Certicom. ECC Challenge. Details available from: `http://www.certicom.com/chal/`

[22] M. Chen and E. Hughes. Protocol failures related to order of encryption and signature: computation of discrete logarithms in RSA groups. *ACISP '98*. Queensland. 1998.

[23] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: Crypto '98*, pages 13–25, 1998.

[24] T. Dierks and B. Anderson. ECC Cipher Suites for TLS. March, 1998. Internet Draft. Available from: `http://www.ietf.org/`

[25] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6): 644–654, November 1976.

[26] W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2: 107–125, 1992.

[27] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31: pages 469–472, 1985.

[28] P. Fahn. IKE authentication using ECDSA. Internet Engineering Task Force, IPSec working group. Internet Draft. March, 2000. Available from: `http://www.ietf.org/`

[29] FIPS 46-2. Data Encryption Standard. *Federal Information Processing Standards Publication 46-2*, 1993. Available from: `http://csrc.nist.gov/`

[30] FIPS 180-1. Secure Hash Standard, *Federal Information Processing Standards Publication 180-1*, 1995. Available from: `http://csrc.nist.gov/`

[31] FIPS 186-2, Digital Signature Standard. *Federal Information Processing Standards Publication 186-2*, 2000. Available from: `http://csrc.nist.gov/`

[32] G. Frey and H.-G. Ruck. A remark concerning *m*-divisibility and the discrete logarithm problem in the divisor class group of curves. *Mathematics of Computation*, 62, pages 865–874, 1994.

[33] FSML. *Financial services markup language*. Financial Services Technology Consortium, August, 1999. Working Draft.

[34] S.D. Galbraith and N.P. Smart. A cryptographic application of the Weil descent. In *Cryptography and Coding*, pages 191-200, 1999.

[35] R. Gallant, R. Lambert, and S.A. Vanstone. Improving the parallelized Pollard lambda search on binary anomalous curves. *Mathematics of Computation*, number 69, pages 1699-1705, 2000.

[36] P. Gaudry, F. Hess, and N.P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Hewlett Packard Laboratories Technical Report*. 2000.

[37] GEC 1. *Test Vectors for SEC 1*. Standards for Efficient Cryptography Group, September, 2000. Working Draft. Available from: `http://www.secg.org/`

[38] D. Harkins and D. Carrel. The Internet Key Exchange. Internet Engineering Task Force, Internet RFC 2409, 1998. Available from: `http://www.ietf.org/`

[39] R. Housley, W. Ford, W. Polk, and D. Solo, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, Internet Engineering Task Force, Internet RFC 2459, 1999. Available from: `http://www.ietf.org/`

[40] IEEE P1363. *Standard Specifications for Public-Key Cryptography*. Institute of Electrical and Electronics Engineers, 2000.

[41] IEEE P1363A. *Standard Specifications for Public-Key Cryptography: Additional Techniques*. May, 2000. Working Draft.

[42] ISO/IEC 14888-3. *Information technology - Security techniques - Digital signatures with appendix - Part 3: Certificate-based mechanisms*.

[43] ISO/IEC 15946-1. *Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 1: General*. 1998. Working draft.

[44] ISO/IEC 15946-2. *Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 2: Digital signatures*. 1998. Working draft.

[45] ISO/IEC 15946-3. *Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 3: Key establishment*. 1998. Working draft.

[46] ITU-T Recommendation X.681. *Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification*. (equivalent to ISO/IEC 8824-2).

[47] ITU-T Recommendation X.690. *Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*. (equivalent to ISO/IEC 8825-1).

[48] D. Johnson. *Diffie-Hellman Key Agreement Small Subgroup Attack, a Contribution to X9F1 by Certicom*. July 16, 1996.

[49] D. Jungnickel. *Finite Fields: Structure and Arithmetics*, B.I.Wissenschaftsverlag, Mannheim, 1993.

[50] B. Kaliski. MQV vulnerability. Posting to ANSI X9F1 and IEEE P1363 newsgroups. 1998.

[51] D. Knuth. *The Art of Computer Programming - Seminumerical Algorithms*, volume 2, Addison-Wesley, second edition 1981.

[52] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48, pages 203–209, 1987.

[53] N. Koblitz. CM-curves with good cryptographic properties. In *Advances in Cryptology: Crypto '91*, pages 279–287, 1992.

[54] N. Koblitz. *A Course in Number Theory and Cryptography*, Springer-Verlag, second edition, 1994.

[55] N. Koblitz, A.J. Menezes, and S.A. Vanstone. The state of elliptic curve cryptography. *Designs, Codes, and Cryptography*, volume 19, pages 173–193, 2000.

[56] P. Kocher, J. Jaffe, B. Jun. Differential power analysis. In *Advances in Cryptology – CRYPTO '99*, pages 388–397, 1999.

[57] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology – CRYPTO '96*, pages 104–113, 1996.

[58] H. Krawcyzk, M. Bellare, and R. Canetti, HMAC: Keyed Hashing for Message Authentication. Internet Engineering Task Force, Internet RFC 2104, 1997. Available from:
`http://www.ietf.org/`

[59] G. Lay and H. Zimmer. Constructing elliptic curves with given group order over large finite fields. *Algorithmic Number Theory*, pages 250–263, 1994.

[60] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. Technical report CORR 98-05, Department of Combinatorics & Optimization, University of Waterloo, March, 1998. Available from:
`http://www.cacr.math.uwaterloo.ca/`

[61] R. Lidl and H. Neiderreiter. *Finite Fields*, Cambridge University Press, 1987.

[62] C.H. Lim and P.J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology: Crypto '97*, pages 249–263, 1997.

[63] R.J. McEliece. *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

[64] A.J. Menezes. *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[65] A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39, pages 1639–1646, 1993.

[66] A.J. Menezes, M. Qu, and S.A. Vanstone. Some new key agreement protocols providing implicit authentication. In *Secondnd Workshop on Selected Areas in Cryptography - SAC '95*, May, 1995.

[67] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.

[68] V. Miller. Uses of elliptic curves in cryptography. In *Advances in Cryptology: Crypto '85*, pages 417–426, 1985.

[69] K. Nyberg and R. Rueppel. A new signature schem based on DSA giving message recovery. *1st ACM Conference on Computer and Communications Security*, pages 58–61, ACM Press, 1993.

[70] K. Nyberg and R. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. *Designs, Codes, and Cryptography*, 7, pages 61–81, 1996.

[71] A. Odlyzko. The Future of Integer Factorization. *CryptoBytes*, volume 1, number 2, pages 5–12, summer 1995.

[72] P. Panjwani and Y. Poeluev. Additional ECC groups for IKE. Internet Engineering Task Force, IPSec working group. Internet Draft. May, 2000. Available from: `http://www.ietf.org/`

[73] S.C. Pohlig and M.E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24, pages 106–110, 1978.

[74] PKCS #8. *Private-key information syntax standard*. RSA Laboratories, November, 1993. Available from: `http://www.rsa.com/rsalabs/pubs/PKCS/`

[75] D. Pointcheval and J. Stern. Security proofs for signatures. *Advances in Cryptology - Eurocrypt '96*, pages 387–398, 1996.

[76] J. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32, pages 918–924, 1978.

[77] T. Satoh and K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Comm. Math. Univ. Sancti Pauli*, number 47, pages 81–92, 1998.

[78] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4, pages 161–174, 1991.

[79] R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44, pages 483–494, 1985.

[80] I. Semaev. Evaluation of discrete logarithm in a group of *P*-torsion points of an elliptic curve in characteristic *P*. *Mathematics of Computation*, number 67, pages 353–356, 1998.

[81] J. Silverman. *The Arithmetic of Elliptic Curves*, Springer-Verlag, 1985.

[82] N. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, volume 12, pages 193–196, 1999.

[83] SEC 2. *Recommended Elliptic Curve Domain Parameters*. Standards for Efficient Cryptography Group, September, 2000. Working Draft. Available from: `http://www.secg.org/`

[84] P.C. van Oorschot and M. Wiener. Parallel collision search with applications to hash functions and discrete logarithms. *2nd ACM Conference on Computer and Communications Security*, pages 210–218, ACM Press. 1994.

[85] P.C. van Oorschot and M. Wiener. On Diffie-Hellman key agreement with short exponents. In *Advances in Cryptology: Eurocrypt '96*, pages 332–343, 1996.

[86] S. Vaudenay. Hidden collisions on DSS. In *Advances in Cryptology: Crypto '96*, pages 83–88, 1996.

[87] WAP WTLS. *Wireless Application Protocol Wireless Transport Layer Security Specification*. Wireless Application Protocol Forum, February, 2000.

[88] WAP WMLScript Crypto API. *Wireless Application Protocol WMLScript Crypto Library Specification*. Wireless Application Protocol Forum, November, 1999.

[89] M. Wiener and R.J. Zuccherato. Fast attacks on elliptic curve cryptosystems. In *Fifth Annual Workshop on Selected Areas in Cryptography: SAC '98*, pages 190–200, 1999.