Standards for Efficient Cryptography

# SEC 1: Elliptic Curve Cryptography

Certicom Research
Contact: Daniel R. L. Brown (dbrown@certicom.com)

Working Draft
February 18, 2005
Version 1.5 1.0

# Contents

**1 Introduction** **1**

1.1 Overview . . . 1
1.2 Compliance . . . 1
1.3 Aim . . . 1
1.4 Document Evolution . . . 2
1.5 Intellectual Property . . . 2
1.6 Organization . . . 2

**2 Mathematical Foundations** **3**

2.1 Finite Fields . . . 3
2.1.1 The Finite Field $\mathbb{F}_p$ . . . 3
2.1.2 The Finite Field $\mathbb{F}_{2^m}$ . . . 4
2.2 Elliptic Curves . . . 6
2.2.1 Elliptic Curves over $\mathbb{F}_p$ . . . 6
2.2.2 Elliptic Curves over $\mathbb{F}_{2^m}$ . . . 8
2.3 Data Types and Conversions . . . 9
2.3.1 Bit-String-to-Octet-String Conversion . . . 9
2.3.2 Octet-String-to-Bit-String Conversion . . . 10
2.3.3 Elliptic-Curve-Point-to-Octet-String Conversion . . . 10
2.3.4 Octet-String-to-Elliptic-Curve-Point Conversion . . . 11
2.3.5 Field-Element-to-Octet-String Conversion . . . 12
2.3.6 Octet-String-to-Field-Element Conversion . . . 13
2.3.7 Integer-to-Octet-String Conversion . . . 13
2.3.8 Octet-String-to-Integer Conversion . . . 14
2.3.9 Field-Element-to-Integer Conversion . . . 14

**3 Cryptographic Components** **15**

3.1 Elliptic Curve Domain Parameters . . . 15
3.1.1 Elliptic Curve Domain Parameters over $\mathbb{F}_p$ . . . 15
3.1.2 Elliptic Curve Domain Parameters over $\mathbb{F}_{2^m}$ . . . 18
3.1.3 Verifiably Random Curves and Base Point Generators . . . 21
3.2 Elliptic Curve Key Pairs . . . 21

3.2.1 Elliptic Curve Key Pair Generation Primitive . . . 22
3.2.2 Validation of Elliptic Curve Public Keys . . . 22
3.2.3 Partial Validation of Elliptic Curve Public Keys . . . 23
3.2.4 Verifiable and Assisted Key Pair Generation and Validation . . . 25
3.3 Elliptic Curve Diffie-Hellman Primitives . . . 25
3.3.1 Elliptic Curve Diffie-Hellman Primitive . . . 26
3.3.2 Elliptic Curve Cofactor Diffie-Hellman Primitive . . . 26
3.4 Elliptic Curve MQV Primitive . . . 27
3.5 Hash Functions . . . 28
3.6 Key Derivation Functions . . . 29
3.6.1 ANSI X9.63 Key Derivation Function . . . 30
3.7 MAC schemes . . . 31
3.7.1 Scheme Setup . . . 32
3.7.2 Key Deployment . . . 32
3.7.3 Tagging Operation . . . 32
3.7.4 Tag Checking Operation . . . 33
3.8 Symmetric Encryption Schemes . . . 33
3.8.1 Scheme Setup . . . 34
3.8.2 Key Deployment . . . 35
3.8.3 Encryption Operation . . . 35
3.8.4 Decryption Operation . . . 35
3.9 Key Wrap Schemes . . . 36
3.9.1 Key Wrap Scheme Setup . . . 36
3.9.2 Key Wrap Schemes Key Generation . . . 37
3.9.3 Key Wrap Schemes Wrap Operation . . . 37
3.9.4 Key Wrap Schemes Unwrap Operation . . . 37
3.10 Random Number Generation . . . 38
3.10.1 Entropy . . . 38
3.10.2 Deterministic Generation of Pseudorandom Bit Strings . . . 38
3.10.3 Converting Random Bit Strings to Random Numbers . . . 39
3.11 Security Levels and Protection Lifetimes . . . 39

**4 Signature Schemes 40**

4.1 Elliptic Curve Digital Signature Algorithm . . . 40
4.1.1 Scheme Setup . . . 41
4.1.2 Key Deployment . . . 41
4.1.3 Signing Operation . . . 41
4.1.4 Verifying Operation . . . 42
4.1.5 Alternative Verifying Operation . . . 43
4.1.6 Public Key Recovery Operation . . . 44
4.1.7 Self-Signing Operation . . . 44

**5 Encryption and Key Transport Schemes 45**
5.1 Elliptic Curve Integrated Encryption Scheme . . . 45
5.1.1 Scheme Setup . . . 46
5.1.2 Key Deployment . . . 47
5.1.3 Encryption Operation . . . 47
5.1.4 Decryption Operation . . . 48
5.2 Wrapped Key Transport Scheme . . . 49

**6 Key Agreement Schemes 51**
6.1 Elliptic Curve Diffie-Hellman Scheme . . . 51
6.1.1 Scheme Setup . . . 51
6.1.2 Key Deployment . . . 52
6.1.3 Key Agreement Operation . . . 53
6.2 Elliptic Curve MQV Scheme . . . 53
6.2.1 Scheme Setup . . . 54
6.2.2 Key Deployment . . . 54
6.2.3 Key Agreement Operation . . . 55

**A Glossary 56**
A.1 Terms . . . 56
A.2 Acronyms . . . 61
A.3 Notation . . . 63

**B Commentary 66**
B.1 Commentary on Section 2 — Mathematical Foundations . . . 66

B.2 Commentary on Section 3 — Cryptographic Components . . . . . 68
B.2.1 Commentary on Elliptic Curve Domain Parameters . . . . . 68
B.2.2 Commentary on Elliptic Curve Key Pairs . . . . . 70
B.2.3 Commentary on Elliptic Curve Diffie-Hellman Primitives . . . . . 70
B.2.4 Commentary on the Elliptic Curve MQV Primitive . . . . . 72
B.3 Commentary on Section 4 — Signature Schemes . . . . . 72
B.3.1 Commentary on the Elliptic Curve Digital Signature Algorithm . . . . . 72
B.4 Commentary on Section 5 — Encryption Schemes . . . . . 74
B.4.1 Commentary on the Elliptic Curve Integrated Encryption Scheme . . . . . 74
B.5 Commentary on Section 6 — Key Agreement Schemes . . . . . 76
B.5.1 Commentary on the Elliptic Curve Diffie-Hellman Scheme . . . . . 76
B.5.2 Commentary on the Elliptic Curve MQV Scheme . . . . . 78
B.6 Alignment with Other Standards . . . . . 80

**C ASN.1** **82**
C.1 Finite Fields . . . . . 82
C.2 Elliptic Curve Domain Parameters . . . . . 84
C.3 Elliptic Curve Public Keys . . . . . 87
C.4 Elliptic Curve Private Keys . . . . . 90
C.5 Signature and Key Establishment Schemes . . . . . 91
C.6 Module . . . . . 94

**D References** **101**

# List of Tables

1 Representations of $\mathbb{F}_{2^m}$ . . . . . 5
5 Computing power required to solve ECDLP . . . . . 67
6 Comparable key sizes . . . . . 69
7 Alignment with other core ECC standards . . . . . 80

# List of Figures

1 Converting between Data Types . . . . . 9

# 1 Introduction

This section gives an overview of this standard.

## 1.1 Overview

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

- signature schemes;
- encryption and key transport 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 Compliance

Implementations may claim compliance witht the cryptographic schemes specified in this document provide the external interface (input and output) to the schemes is equivalent 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.

## 1.3 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 [ANS98b], and WAP WTLS[WAP99], and draft standards like ANSI X9.63 [ANS01a] and IEEE P1363 [IEE00][IEE99], 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.

## 1.4 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. The document is now undergoing its first scheduled review.

This current draft, version 1.5, is a work in progress and is subject to change without notice. Its purpose to seek comment from the SECG.

Changes from Version 1.0 have been manually marked in this version of the document. Added text is marked in red; deleted text is marked in cyan; remaining text is marked in black. Some exceptions may apply to this pattern: headers, footers, citations, references, cross-references, section numbers, page numbers, and this paragraph.

Additional intermittent reviews may also be performed from time-to-time as deemed necessary by the Standards for Efficiency 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 and key transport 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

This section gives an overview of the mathemtical foundations necessary elliptic curve cryptogarphy.

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 [NIS99].

### 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 $a, b \in \{0, 1\}$, see §2.2.) 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 [NIS99]. Composite $m$ was avoided to align this specification with other standards efforts and to address concerns expressed by some experts about the security of elliptic curves defined over $\mathbb{F}_{2^m}$ with $m$ composite — see, for example, [GS].

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 tsrinomial with $k$ as small as possible; otherwise use the degree $m$ binary irreducible pentanomial:

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

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 $\mathcal{O}$ 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:

$$\mathcal{O} + \mathcal{O} = \mathcal{O}.$$

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

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

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

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

— 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 $\mathcal{O}$ 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:

$$\mathcal{O} + \mathcal{O} = \mathcal{O}.$$

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

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

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

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

— 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.

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.

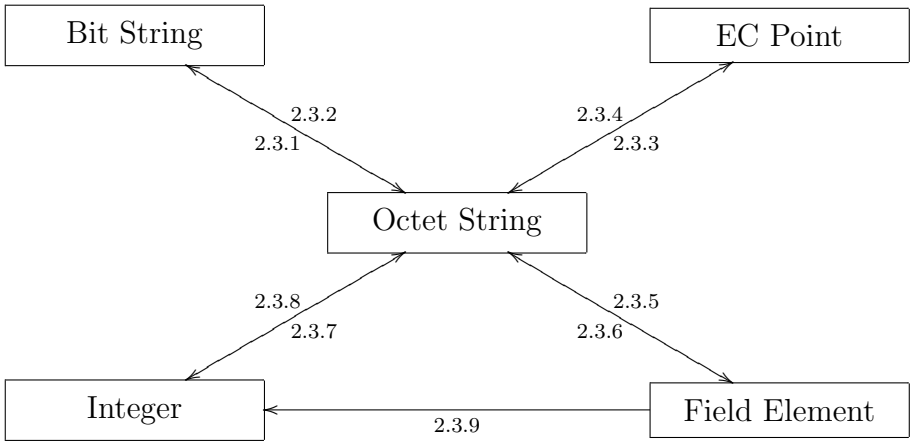


Figure 1: Converting between Data Types

### 2.3.1 Bit-String-to-Octet-String 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 Octet-String-to-Bit-String 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 Elliptic-Curve-Point-to-Octet-String 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 $P = \mathcal{O}$, $mlen = \lceil (\log_2 q)/8 \rceil + 1$ if $P \neq \mathcal{O}$ and point compression is used, and $mlen = 2\lceil (\log_2 q)/8 \rceil + 1$ if $P \neq \mathcal{O}$ 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 $P = \mathcal{O}$, output $M = 00_{16}$.

2. If $P = (x_P, y_P) \neq \mathcal{O}$ 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. Assign the value $02_{16}$ to the single octet $Y$ if $\tilde{y}_P = 0$, or the value $03_{16}$ if $\tilde{y}_P = 1$.

2.4. Output $M = Y \,\|\, X$.

3. If $P = (x_P, y_P) \neq \mathcal{O}$ 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 Octet-String-to-Elliptic-Curve-Point 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 $P = \mathcal{O}$.

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 $\beta$ of $\alpha$ modulo $p$. Output 'invalid' and stop if there are no square roots of $\alpha$ 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 Field-Element-to-Octet-String 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:

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

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 Octet-String-to-Field-Element 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 $M = M_0M_1\ldots M_{mlen-1}$ with $M_i = M_i^0M_i^1\ldots M_i^7$ 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 $a = a_{m-1}x^{m-1} + \cdots + a_1x + a_0$ 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-Octet-String 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 $M = M_0M_1\ldots M_{mlen-1}$ as follows:

1. For $0 \le i \le mlen - 1$, set:

$$M_i = x_{mlen-1-i}.$$

2. Output $M$.

### 2.3.8 Octet-String-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 $M = M_0 M_1 \dots M_{mlen-1}$ to an integer $x$ as follows:

1. View $M_i$ as an integer in the range $[1, 256]$ and set:

$$x = \sum_{i=0}^{mlen-1} 2^{8(mlen-1-i)} M_i.$$

2. Output $x$.

### 2.3.9 Field-Element-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. $a = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_1 x + a_0$. 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 $\mathbb{F}_{2^m}$.

Elliptic curve domain parameters can be verifiably random, which means that the parameters are obtained in part as the output of a secure hash function, applied to some seed value $S$. Verifiably random elliptic domain parameters are recommended, but others may be used for various reasons, such as superior performance.

### 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.

If the elliptic curve domain parameters $T$ are verifiably random, then they should be accompanied by the seed value $S$ from which they are derived.

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 $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$. Optionally, a seed value $S$.

**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 requires 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 $80 < t <\neq 256$, and such that $\lceil \log_2 p \rceil = 521$ if $t = 256$, and such that $\lceil \log_2 p \rceil = 192$ if $t = 80$ 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 $1 \leq B < 100$ 20.
   - $h \leq 2^{t/16}$ 4.
   - $n - 1$ should have a large prime factor.

   If seed $S$ is provided, then the coefficietns $a$ andd $b$, or the point $G$ should be derived from $S$, or both.

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 GSEC 2 [SEC00][SEC99a]. 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. Entity $U$ 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. Entity $U$ 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. Entity $U$ receives assurance in an authentic manner that a party trusted with respect to entity $U$'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. Entity $U$ receives assurance in an authentic manner that a party trusted with respect to entity $U$'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 entity $U$ 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. Optionally, a seed value $S$.

**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 $h \leq 2^{t/16}$ 4, and that $h = \lfloor(\sqrt{p}+1)^2/n\rfloor$.
7. Check that $nG = \mathcal{O}$.
8. Check that $q^B \not\equiv 1 \pmod{n}$ for any $1 \leq B < 100$ 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-Rüuck 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 [ANS05b][ANS98b], it may also be checked that $a$ and $b$ have been correctly derived from the random seed $S$ or that $G$ has been correctly derived from the seed $S$, or all have.

### 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.

If the elliptic curve domain parameters $T$ are verifiably random, then they should be accompanied by the seed value $S$ from which they are derived.

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 $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$. Optionally, a seed value $S$.

**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 requires 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:

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

   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 $1 \leq B < 100$ 20.
   - $h \leq 2^{t/16}$ 4.
   - $n - 1$ should have a large prime factor.

   If seed value $S$ is provided, then coefficients $a$ and $b$, or point $G$ should be derived from it, or both.

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 GSEC 2 [SEC00][SEC99a]. 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. Entity $U$ 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. Entity $U$ 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. Entity $U$ receives assurance in an authentic manner that a party trusted with respect to entity $U$'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. Entity $U$ receives assurance in an authentic manner that a party trusted with respect to entity $U$'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 $h \leq 2^{t/16}$ 4, and that $h = \lfloor(\sqrt{2^m} + 1)^2/n\rfloor$.

8. Check that $nG = \mathcal{O}$.

9. Check that $2^{mB} \not\equiv 1 \pmod{n}$ for any $1 \leq B < 100$ 20, and that $nh \neq 2^m$.

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

Step 9 above excludes the known weak classes of curves which are susceptible to either the Menezes-Okamoto-Vanstone attack, or the Frey-Rüuck 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 [ANS05b][ANS98b], it may also be checked that $a$ and $b$ have been correctly derived from the random seed, and it may also checked that $G$ has been correctly derived from $S$.

### 3.1.3 Verifiably Random Curves and Base Point Generators

The section specified how to derive from a seed $S$ the elliptic curve coefficients $a$ and $b$, and the base point generator $G$. These methods are consistent with ANS X9.62 [ANS05b].

#### 3.1.3.1 Curve Selection

Subject to revision. Preliminary summary.

Apply the hash function to $S$. The resulting output is converted to a $j$-invariant. The $j$-invariant is used to determine $a$ and $b$. For a given $j$, there is some freedom in the selection of $a$ and $b$. Generally, $a$ is chosen to make the ellitpic curve operations slightly more efficient.

#### 3.1.3.2 Point Selection

Subject to revision. Preliminary summary.

Apply the hash function to $S$. Convert the bit string to a field element and an extra bit. The result is converted to a compressed point and then decompressed. If decompression succeeds, cofactor multiplication is used to obtain a point of correct order.

A counter is the input of the hash. It is incremented when the decompression fails, or when cofactor multipliation yields the point at infinity.

## 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. Entity $U$ performs validation of the elliptic curve public key itself using the public key validation primitive described in Section 3.2.2.1.
2. Entity $U$ generates the elliptic curve public key itself using a trusted system.
3. Entity $U$ 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. Entity $U$ 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 [ANS01a], $U$ 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 $Q \neq \mathcal{O}$.
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 $nQ = \mathcal{O}$.
5. If any of the checks fail, output 'invalid', otherwise output 'valid'.

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

In Step 4, it may not be necessary to compute the point $nQ$. For example, if $h = 1$, then $nQ = \mathcal{O}$ is implied by the checks in Steps 2 and 3, because this property holds for any point $Q \in E$. If $h = 2$ and $T$ represents elliptic curve domain parameters over $\mathbb{F}_{2^m}$, then it sufficices to check that the trace of $x_Q$ is 1. A similar computation may be done in other situations where $h$ is small.

### 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. Entity $U$ 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. Entity $U$ generates the elliptic curve public key itself using a trusted system.
3. Entity $U$ 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. Entity $U$ 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 $Q \neq \mathcal{O}$.
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, sSteps 1, 2, and 3 check that $Q$ is a point on $E$ other than the point at infinity.

### 3.2.4 Verifiable and Assisted Key Pair Generation and Validation

Subject to revision.

In certain situations an authority may wish to contribute to the generation of an entity $U$'s key pair. For example, to be certain that entity $U$ has not stolen or fabricated the key pair for a malicious purpose such as identity theft or an unknown key share attack, an authority may give some input into the key pair. As another example, if entity $U$ is not able to provide sufficient entropy into the private key, the authority may wish to supplement the entropy while in a secure environment.

A self-signed signature is a signature in which the message signed contains the signature. It is possible to generate a self-signed ECDSA signature. This is done by selecting by first selected the signature, then the rest of the message, and finally the key pair.

In the case of ECDSA, a self-signed signature ensures a unique key pair per message signed. The function from a self-signed signature to a key pair is essentially one-way, so it is difficult to produce a self-signture that produces a given key pair.

If an authority contributes unique information to message signed, then the authority ensures that the key pair is unique. A unique key pair ensures that the key pair is not somebody else's key pair and that the key pair was not specifically created as part of an attack. (Such as an unknown key share attack.)

An authority can also contribute entropy to the key pair generation by providing some entropy in the message information to be signed. Inclusion of the information in the self-signed signature ensures to the authority the entropy provided was employed in the key pair generation. A reason to do this is if the key pair generator does not have very reliable entropy generation. In that case, the authority can assist the key pair generator. In this situtation, however, to protect the security of the key pair, the authority must be trusted and the self-signed message must be kept confidential.

## 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 simultaneously 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 valid 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 $P = (x_P, y_P) = d_U Q_V$.
2. Check that $P \neq \mathcal{O}$. If $P = \mathcal{O}$, output 'invalid' and stop.
3. Output $z = x_P$ as the shared secret field element.

### 3.3.2 Elliptic Curve Cofactor Diffie-Hellman Primitive

Entity $U$ 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 partially valid 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 $P \neq \mathcal{O}$. If $P = \mathcal{O}$, 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 simultaneously 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 partially valid 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 $P \neq \mathcal{O}$. If $P = \mathcal{O}$, output 'invalid' and stop.

7. Output $z = x_P$ as the shared secret field element.

## 3.5 Hash Functions

Subject to revision.

Days before the release date, an attack was announced that finds a collision in SHA-1 in about $2^{69}$ hash operations.

This decreases the security of SHA-1 against collision resistance. Collision resistance is most important for ECDSA in this standard, because it is necessary to resist exisential forgery by an active chosen message attack.

In situations where ECDSA with SHA-1 is used and 80 bits of security against existential forgery by active chosen messsage attacks is necessary, then some countermeasures may be recommended in a future revision of this document.

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-224
SHA-256
SHA-384
SHA-512

These hash functions are SHA-1 is specified in FIPS 180-21 [FIP04][FIP95]. They It maps octet strings of length less than a certain number of $2^{61}$ octets to hash values which are octet strings of a fixed 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
IKEv2-KDF
TLS-KDF

The key derivation function ANSI-X9.63-KDF is the simple hash function construct described in ANSI X9.63 [ANS01a]. 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.

The key derivation functions IKEv2-KDF and TLS-KDF may only be used with the elliptic curve Diffie-Hellman scheme for use in the IVEv2 and TLS protocols, respectively. The function IKEv2-KDF is specified in [HC98] and [Kau05]. The function TLS-KDF is specified in [DA99] and [GBWM+04].

### 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
HMAC–SHA-224–112 with 28 octet or 224 bit keys
HMAC–SHA-224–224 with 28 octet or 224 bit keys
HMAC–SHA-256–128 with 32 octet or 256 bit keys
HMAC–SHA-256–256 with 32 octet or 256 bit keys
HMAC–SHA-384–192 with 48 octet or 384 bit keys
HMAC–SHA-384–284 with 48 octet or 384 bit keys
HMAC–SHA-512–256 with 64 octet or 512 bit keys
HMAC–SHA-512–512 with 64 octet or 512 bit keys

The first two of Both these MAC schemes are specified in IETF RFC 2104 [KBC97] and ANSI X9.71 [ANS01b] based on the hash function SHA-1 specified in FIPS 180-1 [FIP95]. Following the notation suggested in [KBC97], 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

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

1. Entities $U$ and $V$ should establish which of the supported MAC schemes to use (and if appropriate select any initial values required by the MAC scheme). 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

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

1. Entities $U$ 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

Entity $U$ 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

Entity $V$ 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 messages from their ciphertexts so that the schemes provide data confidentiality.

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

3-key TDES in CBC mode
XOR encryption scheme
AES-128 in CBC mode
AES-192 in CBC mode
AES-256 in CBC mode

The block cipher 3-key TDES in CBC mode is specified in ANSI X9.52 [ANS98a]. 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.

The block cipher AES is specified in [FIP01b]. The CBC mode of AES is specified in [NIS01b].

Furthermore here the 8 octet or 64 bit $IV$ for TDES in CBC mode should always take the value $00000000_{16}$. Furthermore here the 16 octet or 128 bit $IV$ for AES in CBC mode should always take the value $0000000000000000_{16}$.

(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.)

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.

The block ciphers 3-key TDES AES in CBC mode is designed to provide semantic security in the presence of adversaries launching chosen-message 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 requirements above apply to ECIES. Other symmetric encryption schemes may be used in elsewhere in the system. In general uses of CBC mode, the IV should be chosen as an unpredictable value. (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 standardsection, 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

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

1. Entities $U$ and $V$ should establish which of the supported symmetric encryption schemes to use (and if appropriate select any initial values required by the encryption scheme). 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

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

1. Entities $U$ 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

Entity $U$ should encrypt messages to send to entity $V$ 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

Entity $V$ should decrypt ciphertext from entity $U$ 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 and a symmetric encryption key $K$.

**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 $M$.

## 3.9 Key Wrap Schemes

This subsection specifies that either the NIST AES key wrap algorithm or the CMS TDES key wrap algorithm are

- must be used as the key wrap schem in the Wrapped Key Agreement Key Transport Scheme, and
- should be used more generally when wrapping an existing symmetric key with another symmetric key.

The AES key wrap algorithm was first specified in [NIS01a]. It has also been re-specified in [SH02]. Currently, ASC X9 is also re-specifying it, with some minor extension for additional input in [ANS03], and has requested public review of the algorithms therein [Dwo04].

The AES key wrap algorithm may be used with the AES block cipher or the TDES block cipher. When using the AES block cipher to wrap keys, the AES key wrap algorithm must be used. When using the TDES block cipher, however, another key wrap algorithm, the CMS TDES key wrap algorithm, may be used for backwards interoperability reasons. This algorithm was first specified [Hou99] and is also being re-specified in [ANS03].

For clarity in the remainder of this standard, 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 3.9.1, the key deployment procedure is specified in Section 3.9.2, the wrap operation is specified in Section 3.9.3, and the unwrap operation is specified in Section 3.9.4.

### 3.9.1 Key Wrap Scheme Setup

Entities $U$ and $V$ should perform the following setup procedure to use a key wrap scheme:

1. Entities $U$ and $V$ should establish which of the supported key wrap schemes to use (and if appropriate select any initial values required by the key wrap scheme). Let *WRAP* denote the encryption scheme chosen, and *wrapkeylen* denote the length in octets of the keys used by the scheme.

### 3.9.2 Key Wrap Schemes Key Generation

Entities $U$ and $V$ should perform the following key deployment procedure to use the key wrap scheme:

1. Entities $U$ and $V$ should establish a key-encryption key $K$ of length *wrapkeylen* octets.

### 3.9.3 Key Wrap Schemes Wrap Operation

Entity $U$ should wrap keys to send to entity $V$ using the key-encryption key and key wrap parameters established during the setup procedure and the key deployment procedure as follows:

**Input:** An octet string $C$ which is the key to be wrapped.

**Output:** An octet string $W$ which is the wrapped key corresponding to $C$, or 'invalid'.

**Actions:** Compute the wrapped key $W$ 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 wrapped key $\overline{W}$ of $\overline{C}$ using the key wrap operation of the selected key wrap scheme under the key-encryption key $\overline{K}$. If the key wrap operation outputs 'invalid', output 'invalid' and stop.
3. Convert $\overline{W}$ to an octet string $W$ using the conversion routine specified in Section 2.3.1.
4. Output the octet string $W$.

### 3.9.4 Key Wrap Schemes Unwrap Operation

Entity $V$ should unwrapped a wrapped key from entity $U$ using the key-encryption key and key wrap parameters established during the setup procedure and the key deployment procedure as follows:

**Input:** A key-encryption key $K$ and an octet string $W$ which is the wrapped key.

**Output:** An octet string $C$ which is the unwrapping of $W$, or 'invalid'.

**Actions:** Unwrap $W$ as follows:

1. Convert $W$ to a bit string $\overline{W}$ and $K$ to a bit string $\overline{K}$ using the conversion routine specified in Section 2.3.2.
2. Calculate the unwrapping $\overline{C}$ of $\overline{W}$ using the unwrap operation of the selected key wrap scheme under the shared key-encryption key $\overline{K}$. If the unwrap 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 $C$.

## 3.10 Random Number Generation

Subject to revision.

Cryptographic keys must be generated in a way that prevents an adversary from guessing the private key. Private keys should be generated with the help of a random number generator.

Random number generators must comply with ANS X8.92 [ANS05a] or any corresponding NIST publication.

For convenience, a subset of the specification are included here. Implementations may optionally choose to comply with this subset, but implementations are not restricted to this subset.

### 3.10.1 Entropy

A random number generator (RNG) maintains a state. The output of the random number generator is a function of the state. The security of the RNG depends on the maximum probability that its state taking any one value. For a security level of $t$ bits, the maximum probability of any state value must be at most $2^{-t}$. Generally, the security level of a cryptographic system is no more than the security level of the RNG from which its cryptographic keys are derived.

Note: When the maximum probability in a probability distributaion is $2^{-t}$, that distribution is said to have *min-entropy* of $t$ bits. Min-entropy is never more than Shannon entropy. Shannon entropy is generally not enough to ensure an adequate security in cryptography, because of pathological probability distributions.

As a precautionary measure against the risk that two different RNG will collide with the same state, an RNG should also be personalized with a value that is not likely to repeated for over $2^{64}$ uses. The personalization value need not be secret. One way to achieve personalization is increase the amount of min-entropy necessary at the instantiation of the RNG to $t + 64$ bits.

### 3.10.2 Deterministic Generation of Pseudorandom Bit Strings

The output of an RNG must be a one-way function of its state to ensure that the state is not revealed. Several one-way function are available. Upon output, the state must be updated, so that the future outputs cannot be different.

The state should be updated with a one-way function, so that past states cannot be learnt from a future compromised state. This attribute is sometimes called forward secrecy or backtracking resistance. Some of the schemes in the standard, such as MQV, provide forward secrecy. When forward secrecy is a security objective of these schemes, then the RNG used must also provide forward secrecy.

In some circumstances, it is necessary that the RNG be able to recover from compromise of the current state. This can only be accomplished by the injection of new entropy. This security attribute is sometimes recoverable security or prediction resistance. This is an optional attribute, for very high security applications.

Some allowed deteriministic random bit generators will be specified.

### 3.10.3 Converting Random Bit Strings to Random Numbers

Elliptic curve private keys are integers in a certain range. For full security, these integers should have a probability distribution that is as close to possible to uniform. (Otherwise, a variety of attacks may become possible.)

The deterministic algorithms in the previous section produced random bit strings. Bit strings can be converted to integers, but the range is not exactly that needed for the elliptic private key. The following process may be used to convert a random bit string to a random integer, in such a way that if the bit string is uniform then so is the integer.

To be added.

## 3.11 Security Levels and Protection Lifetimes

Data protected with cryptography today may continue to need protection in the future. Advances in cryptanalysis can be predicated, at least approximately.

Based on current approximations, this document requires that data that needs protection beyond the year 2010 must be protected with 112-bit security or higher. Data that needs protection beyond the year 2030 must be protected with 128-bit security or higher.

# 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

Entities $U$ 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 $U$.

Entity $V$ may also wish to must 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

Entities $U$ 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 $U$.

Entity $V$ may wish to must 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

Entity $U$ 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

Entity $V$ should verify signed messages from entity $U$ 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 $R = \mathcal{O}$, 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'.

### 4.1.5 Alternative Verifying Operation

A signer $U$ may verify $U$'s own signatures more efficiently with the following operation, which use $U$'s private key.

A situation where this could be useful is a CA that verifies its own certificates.

Details to be added. For now, a brief summary follows. All verification steps are the same, except that in Step 5, the verifier instead computes

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

The benefit of this is that the verifier needs just a single scalar multiplication.

### 4.1.6 Public Key Recovery Operation

Subject to revision. Details to be added.

Given an ECDSA signature $(r, s)$ and EC domain parameters, it is generally possible to determine the public key $Q$, at least to within a small number of choices.

This is useful for generating self-signed signatures.

This is also useful in bandwidth constrained environments, when transmission of public keys cannot be afforded. Entity $U$ could send a signature to entity $V$, who recovers $Q_U$. Entity $V$ can look up the public key in some certificate or directory, and if it matches then the signature can be accepted. Alternatively, entity $U$ may transmit the signature together with the certificate except the public key is omitted from the certificate. For example, in long certificate chains signed with ECDSA, bandwidth can be saved by omission of the public keys.

Potentially, several possible public keys can be recovered from a signature. At a small cost, the signer can generate the ECDSA signature in such a way that only one of the potential public keys is viable, and such that the verifier has a very small additional cost of determining which is the correct public key.

### 4.1.7 Self-Signing Operation

Subject to revision. Details to be added.

To generate a self-signed ECDSA signature the following operation can be used.

Select a random signature value $(r, s)$. Form a message $M$ containing a copy $(r, s)$.

Use the Public Key Recovery Operation in §4.1.6 to recover a public key $Q$.

# 5 Encryption and Key Transport Schemes

This section specifies the public-key encryption and key transport 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 $M$.

Key transport schemes are a special class of public-key encryption schemes where the message $M$ is restricted to be a cryptographic key, usually a symmetric key. Except for this restriction, most of the discusssion below about to public-key encryption schemes also applies to key transport schemes.

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 messages from their ciphertexts so that 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 encryptions schemes supported at this time is are the Elliptic Curve Integrated Encryption Scheme (ECIES) and the the general construction of combining a key agreement scheme with a key wrap mechanism. The first, ECIES, is specified in Section 5.1. The second general construction is specified in Section 5.2.

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 both semantically secure and plaintext-aware , in the presence of an adversary capable of launching chosen-plaintext and chosen-ciphertext attacks. Note that

the Elliptic Curve Integrated Encryption Scheme or ECIES is unrelated to the AES symmetric 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

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

1. Entity $V$ 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. Entity $V$ 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. Entity $V$ 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. Entity $V$ 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. Entity $V$ should establish EC 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. Entity $U$ 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. Entity $U$ should establish whether or not to represent elliptic curve points using point compression.

8. Entities $U$ and $V$ should establish an expected format of $SharedInfo_2$ that is suffix-free.

### 5.1.2 Key Deployment

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

1. Entity $V$ 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. Entity $U$ 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 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 $M$.

## 5.2 Wrapped Key Transport Scheme

To be revised. Follows NIST SP 800-56 [NIS03] and RFC 3278 [BWBL02].

The wrapped key transport scheme uses a combination of a key wrap scheme and a key agreement scheme. The key agreement used can be either Diffie-Hellman (§6.1) or MQV (§6.2), but in either case it must be a 1-pass variant. In a 1-pass variant of a key agreement scheme, in the key deployment phase, entity $U$ must obtain authentic copies of all of the keys of $V$, in addition to the usual key deployment operations. For Diffie-Hellman, entity $U$ must additionally obtain $Q_V$ in an authentic manner. For MQV, entity $U$ must additionally obtain $Q_{2,V}$ in an authentic manner, which may be achieved most easily if $Q_{2,V} = Q_{1,V}$, which is the default choice in the absent of any indication otherwise.

Once $U$ has obtained all of the keys of $V$ in an authentic manner, such as by extracting them from a certificate, then to complete key deployment entity $U$ needs only to send any of its remaining keys to entity $V$. Hence, all ephemeral keys are exchanged in a single pass.

In wrapped key tranport, entity $U$ uses a 1-pass key agreement operation with entity $V$ to agree on a key $K$ and then wraps a key $C$ with $K$ to obtain a wrapped key $W$. The wrapped key $W$ is sent together with the public keys of $U$ in the single pass of the exchange.

Any format for combining the public keys and wrapped keys into a single pass message are allowed, including the formats used in S/MIME [BWBL02]. For convenience, this will include a pre-defined format that may find use in future applications.

A typical application of the transport key $C$ is for encrypting and authenticating content data, in a single pass. The key $C$ is often called a content-encryption key. Generally, the encrypted message and authentication tag, or both, will be sent in a single pass together with wrapped key and any necessary public keys. The most familiar single pass application is email.

If entity $U$ wraps a single key $C$ for many different recipients, which is useful for protecting an email sent to many recipients, say $V_1, V_2, \ldots$, then $U$ may re-use the same ephemeral public key with for each $V_i$. We denote $K_i$ by the key agreed between $U$ and $V_i$, and $W_i$ the wrapping of $C$ with $K_i$.

In this situation, entity $V_i$ learns the key $C$. This brings a risk that entity $V_i$ will abuse $C$ against another recipient $V_j$ to alter the message and make $V_j$ think the altered message came from $U$.

Entity $U$ often wish to prevent this problem. If $M$ is the message authenticated and $T = MAC_C(M)$ is the MAC tag used to authenticated. Entity $U$ may include $T$ as part of the *SharedInfo* used in the KDF with key agreement operation. When this is done, any modification to the message will modify $T$, which will modify $K_j$, which will modify $W_j$. Entity $V_i$ will not be able to produce the correctly modified $W_j$, so if entity $V_i$ modifies the message, then entity $V_j$ will not re-compute the key $C$ correctly and the message authentication will fail.

Alternatively, entity $U$ may include $T$ as an optional parmater intot the key wrap scheme. This has a similar effect. Yet another option is for entity $U$ to compute a separate tag $T_i$ for each recipient. If the message is very long and the number of recipients is large, then computing many MAC tags on a long message is very slow. For a faster approach, entity $U$ can instead compute $T_i = MAC_{K_i}(T)$, where $T$ is as before.

# 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

Entities $U$ 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

Entities $U$ 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

Entities $U$ 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.

Entity $U$ 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

Entities $U$ 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

Entities $U$ 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 partially valid using one of the methods specified in Section 3.2.2 or Section 3.2.3.

### 6.2.3 Key Agreement Operation

Entities $U$ 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.

Entity $U$ 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 the certificate 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. 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 $\mathcal{O}$ 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 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 any pre-specified output, and it is computationally infeasible to find any two 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 keying data suitable for subsequent cryptographic use by cryptographic schemes to its legitimate users. |
| 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 $\mathcal{O}$. 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 $nP = \mathcal{O}$ (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. |
| plaintext-awareness | An encryption scheme is plaintext-aware if it is infeasible to generate a valid ciphertext without knowing the corresponding message. |
| 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 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 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. See [FIP01b]. |
| ANS | American National Standard. |
| ANSI | American National Standards Institute. |
| ASC X9 | Accredited Standards Committee X9. |
| ASN.1 | Abstract Syntax Notation One. |
| CA | Certification Authority. See [BHP02]. |
| CMS | Cryptographic Message Syntax. See [Hou99]. |
| CRL | Certificate Revocation List. See [BHP02]. |
| DER | Distinguished Encoding Rules. See [ITUb]. |
| DES | Data Encryption Standard. See [FIP93b]. |
| DSA | Digital Signature Algorithm. See [FIP01a][FIP93a]. |
| DSS | Digital Signature Standard. See [FIP01a]. |
| EC | Elliptic Curve. |
| ECC | Elliptic Curve Cryptography. |
| ECIES | Elliptic Curve Integrated Encryption Scheme. See Section 5.1. |
| ECDH | Elliptic Curve Diffie-Hellman. See Section 6.1. |

| | |
|---|---|
| ECDLP | Elliptic Curve Discrete Logarithm Problem. |
| ECDSA | Elliptic Curve Digital Signature Algorithm. See Section 4.1. |
| ECMQV | Elliptic Curve Menezes-Qu-Vanstone. See Section 6.2. |
| FIPS | Federal Information Processing Standard. |
| FSML | Financial Services Markup Language. |
| FSTC | Financial Services Technology Consortium. |
| GEC | Guideline for Efficient Cryptography. |
| HMAC | Hash-based Message Authentication Code. See [KBC97]. |
| IACR | International Association for Cryptologic Research |
| IEC | International Electrotechnical Commission. |
| IEEE | Institute of Electrical and Electronics Engineers. |
| IETF | Internet Engineering Task Force. |
| IKE | Internet Key Exchange. |
| ISO | International Organization for Standardization. |
| ITU | International Telecommunications Union. |
| LNCS | Lecture Notes in Computer Science |
| KDF | Key Derivation Function. |
| MQV | Menezes-Qu-Vanstone. See [LMQ$^+$98]. |
| NIST | (U.S.) National Institute of Standards and Technology. |
| PKI | Public Key Infrastructure. |
| PKIX | Public Key Infrastructure for the Internet. |
| RFC | Request for Comment. |
| RNG | Random Number Genrator. |
| SEC | Standard for Efficient Cryptography |
| SHA-1 | Secure Hash Algorithm Revision One. See [FIP95]. |
| SSL | Secure Sockets Layer. |
| TDES | Triple Data Encryption Standard. See [ANS98a]. |
| TLS | Transport Layer Security. |
| WAP | Wireless Application Protocol. |
| WTLS | Wireless Transport Layer Security. |

## A.3 Notation

The notation adopted in this document is:

| | |
|---|---|
| $[X]$ | 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 $\lfloor 5.3 \rfloor = 6$. |
| $x \bmod n$ | 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 $\mathcal{O}$. |
| $\#E(\mathbb{F}_q)$ | 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 $\mathcal{O}$). $\#E(\mathbb{F}_q)$ 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 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. |

| | |
|---|---|
| $\mathrm{mod}\, f(x)$ | Arithmetic modulo the polynomial $f(x)$. |
| $\mathrm{mod}\, n$ | Arithmetic modulo $n$. |
| $n$ | The order of the base point $G$. |
| $\mathcal{O}$ | 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. |
| $\|\|X\|\|$ | Length in octets of the octet string $X$. |
| $X \,\|\|\, Y$ | 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 to be updated further. Current updates are preliminary and very limited.

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 February 2005 August 1999. 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 [Jun93], Lidl and Neiderreiter [LN87], and McEliece [McE87]. An introduction to elliptic curves can be found in the book of Silverman [Sil85].

Elliptic curves over finite fields were first proposed for use to build cryptographic schemes in 1985 by Koblitz [Kob87] and Miller [Mil92]. Excellent treatments focusing on ECC are contained in Blake, Seroussi, and Smart [BSS99], Koblitz [Kob94], Koblitz, Menezes, and Vanstone [KMV], and Menezes [Men93].

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-$\rho$ method and the Pollard-$\lambda$ method [Pol78]. The Pollard-$\rho$ method takes about $\sqrt{\pi n/2}$ steps, and the Pollard-$\lambda$ method takes about $2\sqrt{n}$ steps. Both methods can be parallelized effectively — see [vOW94].

Gallant, Lambert, and Vanstone [GLV], and Wiener and Zuccherato [WZ99] recently showed that the Pollard-$\rho$ method can be sped up by a factor of $\sqrt{2}$. Thus the expected running time of the

Pollard-$\rho$ 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-$\rho$ 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-$\rho$ 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 5 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-$\rho$ method. To place Table 5 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 [Odl95]. Table 5 is reproduced from ANSI X9.62 [ANS05b][ANS98b]. More details on how the estimates were obtained can be found there.

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

Table 5: Computing power required to solve ECDLP

The difficulty of the ECDLP is further illustrated by van Oorschot and Wiener's 1994 paper [vOW94]. 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 $n \approx 2^{120}$ in about 35 days. Hardware attacks on larger values of $n$, like $n \approx 2^{160}$, appear completely infeasible at this time.

Finally, although no general subexponential algorithms to solve the ECDLP are known, threetwo 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 [MOV93], and Frey and Rüuck [FR94]. 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 [Sem], Smart [Sma], and Satoh and Araki [SA]. This attack efficiently maps the elliptic curve $E$ into the additive group of $\mathbb{F}_q$. Thirdly, for curves defined over $\mathbb{F}_q$ where $q = 2^m$ with $m$ composite, certain attacks based on Weil descent, more recently, index calculus, have been discovered. This is an ongoing research area. On a precautionary basis, however, such curves should be avoided. Both tThese weak classes of curves are excluded in this document.

Additional information on the difficulty of the ECDLP can be found in ANSI X9.62 [ANS05b][ANS98b],

ANSI X9.63 [ANS01a], Blake, Seroussi, and Smart [BSS99], and Koblitz, Menezes, and Vanstone [KMV]. A useful source of information on the state-of-the-art in practical attacks on the ECDLP is Certicom's ECC challenge [ECC].

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 in, for example, [Knu81, McE87, MvOV97]. A variety of efficient general techniques for computing point scalar multiplication are known and exploit tricks like switching to projective co-ordinates and precomputation.

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 [ABMV93, AMV93, AMOV91, NIS99]. Examples of elliptic curves amenable to particularly efficient implementation include Koblitz curves over $\mathbb{F}_{2^m}$ [Kob92] 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 [BSS99], and Koblitz, Menezes, and Vanstone [KMV].

## 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 [BDR+96] provides guidance which may help implementers make this determination. In particular, the authors of [BDR+96] 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 6 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.

| Security level (in bits) | Symmetric scheme (key size in bits) | ECC-based scheme (size of $n$ in bits) | DSA/RSA (modulus size in bits) | Protects to year |
|---|---|---|---|---|
| 56 | 56 | 112 | 512 | 2000 |
| 80 | 80 | 160 | 1024 | 2010 |
| 112 | 112 | 224 | 2048 | 2030 |
| 128 | 128 | 256 | 3072 | 2030+ |
| 192 | 192 | 384 | 7680 | 2030+ |
| 256 | 256 | 512 | 15360 | 2030+ |

Table 6: 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 [Sch85], or one of its various improvements, 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' [LZ94].
- 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 [ANS05b][ANS98b].

SEC 2 [SEC00] GEC 1 [SEC99a] — 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 [ANS05b][ANS98b], ANSI X9.63 [ANS01a], and IEEE P1363 [IEE00][IEE99].

### 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 [ANS01a].

### 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, for example, that if the ECDLP is easy then so is the ECDHP. Boneh and Lipton [BL96] 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 [BV96, Bon98].

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 [Joh96, LL97] 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 [ANS01a].

### 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 [LMQ+98] for a discussion of this conjecture.

Additional information on the elliptic curve MQV primitive can be found in ANSI X9.63 [ANS01a].

## 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 [FIP93a]. It was first proposed in ANSI X9.62 [ANS05b][ANS98b].

The ECDSA was chosen for inclusion in this document because it is widely standardized in, for example, ANSI X9.62 [ANS05b][ANS98b], IEEE P1363 [IEE00][IEE99], and ISO 15946-2 [ISO98b]. It's 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 [Sch91], which was shown to be provably secure in the random oracle model based on the ECDLP in [PS96], or the Nyberg-Rueppel scheme [NR93, NR96], 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 [PH78]. $U$ 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 [BWM99, CH98] 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 [BWM99, CH98]. Another class of attack to avoid is described in [ABM+03]. 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 [Vau96] 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 [BDL97], power-analysis attacks [KJJ99], and timing-analysis attacks [Koc96].

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 [ANS05b][ANS98b] and IEEE P1363 [IEE00][IEE99]. Test vectors for ECDSA can be found in SEC 2 [SEC00] GEC 2 [SEC99b].

## 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 both semantically secure and plaintext-aware, in the presence of an adversary capable of launching chosen-plaintext and chosen-ciphertext attacks. It was proposed in [ABR98, BR97].

(Note that the specification of ECIES here differs slightly from the description in [ABR98] 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 [ABR98] 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 [ABR98].)

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 [ABR98]. 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 [ANS01a].

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 [ElG85] does not offer security against chosen-ciphertext attacks, while the elliptic curve analog of the Cramer-Shoup encryption scheme [CS98] 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 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 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 [PH78]. $V$ 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 [Joh96, LL97] 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 [BDL97], power-analysis attacks [KJJ99], and timing-analysis attacks [Koc96].

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 the need to balance the added security offered by the former against the bandwidth savings offered by the latter. 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 [ABR98], 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 [ANS01a], and the paper of Abdullah, Bellare, and Rogaway [ABR98]. Test vectors for ECIES can be found in GEC 2 [SEC99b].

## 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 [DH76]. It was first proposed in [DH76, Kob87, Mil92].

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 [ANS01a], IEEE P1363 [IEE00][IEE99], and ISO 15946-3 [ISO98c]. Examples of the application of the elliptic curve Diffie-Hellman scheme to achieve a variety of security goals can be found in [BCK98, BWJM97, DvOW92]. 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 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 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 [PH78]. $U$ 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 [Joh96, LL97], 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 [BDL97], power-analysis attacks [KJJ99], and timing-analysis attacks [Koc96].

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 [ANS01a], and IEEE P1363 [IEE00][IEE99]. Test vectors for the elliptic curve Diffie-Hellman scheme can be found in GEC 2 [SEC99b].

### 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 [LMQ+98, MQV95].

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 [ANS01a], and IEEE P1363 [IEE00][IEE99].

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 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 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 [PH78]. $U$ and $V$ should therefore receive assurance that the elliptic curve domain parameters used are valid.

- Unknown key-share attacks. Kaliski [Kal98] 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 [BDL97], power-analysis attacks [KJJ99], and timing-analysis attacks [Koc96].

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 [ANS01a], in IEEE P1363 [IEE00][IEE99], and in the paper of Law, Menezes, Qu, Solinas, and Vanstone [LMQ+98]. Test vectors for the elliptic curve MQV scheme can be found in GEC 2 [SEC99b].

## 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 [ANS05b][ANS98b], ANSI X9.63 [ANS01a], IEEE P1363 [IEE00][IEE99], IPSec [HC98], ISO 14888-3 [ISO], and ISO 15946 [ISO98a, ISO98b, ISO98c]. Table 7 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 |
| IEEE P1363 | ECDSA, ECDH, ECMQV |
| IPSec | ECDH |
| ISO 14888-3 | ECDSA |
| ISO 15946 | ECDSA, ECDH, ECMQV |

Table 7: 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 draft IEEE P1363 standard has a wide scope encompassing schemes based on the integer factor-

ization 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 ECSSA), 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. ECIES has been submitted for inclusion in the proposed addendum to IEEE P1363, IEEE P1363A [IEE04][IEE97].

The IPSec standards include 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.

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 [Fin99] allows ECDSA as specified in ANSI X9.62 to be used to sign electronic checks and other electronic financial documents. The PKIX ECDSA standard [BJP99] specifies how to use ECDSA as specified in ANSI X9.62 within the PKIX certification framework. The SSL/TLS ECC standard [DA98] 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 [WAP99] 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. 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 PKIX ECDSA standard, the SSL/TLS ECC standard, and the WAP WTLS standard.

# C ASN.1

This section specifies suggests the ASN.1 syntax that which may should be used when ASN.1 syntax is used to convey parts of this information. when Generally, the ASN.1 syntax needs to be suitably encoded via, for example, DER [ITUb]. Several different types of information may need to be conveyed during the operation of the schemes specified in this document. 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 **??** recommends syntax to describe signature and key establishment schemes. Section C.6 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 profiles follows the syntax used in ANSI X9.62 [ANS05b][ANS98b] and PKIX [HFPS99], [BJP99], [BHP02] and [Bro04].

## C.1 Finite Fields

This section provides the recommended ASN.1 syntax to identify finite fields and specific elements of said fields. 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 [ANS05b][ANS98b, 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 [ITUa, 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 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, as a choice of three parameters, is recommended (following [BJP99]) for use in X.509 certificates and elsewhere.

```
ECDomainParameters{ECDOMAINCURVES:IOSet} ::= CHOICE {
     specifiedecParameters SpecifiedECDomainECParameters,
     namedCurve CURVESECDOMAIN.&id({IOSet}),
     implicitCA NULL
}
```

The choice of three parameters (above) allows detailed specification of all required values using the choice `ecParameters`, the use of a `specifiednamedCurve` as an object identifier substitute for a particular set of elliptic curve domain parameters, or `implicitCA` to indicate that the parameters are explictly defined elsewhere. The valid values for the `namedCurve` choice are constrained to those

within the class ECDOMAINCURVES (defined below and explained further in SEC 2 [SEC00]GEC 1 [SEC99a] ).

The following syntax is used to describe explicit representations of elliptic curve domain parameters, if need be. The inclusion of the cofactor is strongly recommended.

```
SpecifiedECDomainECParameters ::= SEQUENCE {
     version SpecifiedECDomainVersionINTEGER { ecpVer1(1) } (ecdpVer1 | ecdpVer2 |
ecdpVer3, ...),
     fieldID FieldID {{FieldTypes}},
     curve Curve,
     base ECPoint,
     order INTEGER,
     cofactor INTEGER OPTIONAL,
     hash HashAlgorithm OPTIONAL,
      ...
}
```

The components of type SpecifiedECDomainParameters have the following meanings.

- The component `version` is the version number of the ASN.1 type with a value of 1, 2 or 3. The notation above creates an INTEGER named ecpVer1 and assigns to it the value one. It The notation above is used to constrain `version` to a single set of values. The meaning of these three values are as follows. Details to be added.
- 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 recommended that that the cofactor be included in order to facilitate interoperability between implementations.
- The component `hash` is the hash function used to generate the domain parameters verifiably at random.

The type SpecifiedECDomainVersion is a subtype of INTEGER and is used to contrain the set of versions.

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

```
}
```

The type `Curve` itself is by the following, namely 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 [ANS05b] [ANS98b] 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
     -- Shall be present if used in SpecifiedECDomain with version equal to ecdpVer2
or ecdpVer3
}
```

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 class `ECDOMAINCURVES`, defined as follows, is used to specify a named curve.

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

For example, the curve `sect163k1`, defined in SEC 2 [SEC00] GEC 1 [SEC99a], is denoted by the syntax `ID sect163k1`.

The following syntax, included here for completeness, may be extended by other standards and implementations to specify the list of supported named curves. One such extension may be found in SEC 2 [SEC00] GEC 1 [SEC99a]; another such extension may be found in ANSI X9.62 [ANS05b][ANS98b].

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

The following type `HashAlgorithm` is used to specify a hash function.

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

The information object set `HashFunctions` specifies the allowed hash functions currently:

```
HashFunctions ALGORITHM ::= {
```

```
    {OID sha-1} | {OID sha-1 PARMS NULL } |
    {OID id-sha224} | {OID id-sha224 PARMS NULL } |
    {OID id-sha256} | {OID id-sha256 PARMS NULL } |
    {OID id-sha384} | {OID id-sha384 PARMS NULL } |
    {OID id-sha512} | {OID id-sha512 PARMS NULL } ,
    ...  -- Additional hash functions may be added in the future }
```

The following object identifiers are used above to identify specific hash functions.

```
sha-1 OBJECT IDENTIFIER ::= {iso(1) identified-organization(3)
oiw(14) secsig(3) algorithm(2) 26}
id-sha OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16) us(840)
organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) }
id-sha224 OBJECT IDENTIFIER ::= { id-sha 4 }
id-sha256 OBJECT IDENTIFIER ::= { id-sha 1 }
id-sha384 OBJECT IDENTIFIER ::= { id-sha 2 }
id-sha512 OBJECT IDENTIFIER ::= { id-sha 3 }
```

## C.3 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 [BJP99]) 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 either to the object ecPublicKeyType (below) or the object ecdsa-SHA1 (as defined in ANSI X9.62 [ANS05b][ANS98b, Section 7.4]). (Additional algorithm identifiers may be added.)

```
ECPKAlgorithms ALGORITHM ::= {
     ecPublicKeyType |
     ecPublicKeyTypeRestricted |
     ecPublicKeyTypeSupplemented ,
     ...
}
ecPublicKeyType ALGORITHM ::= {
     OID id-ecPublicKey PARMS ECDomainParameters {{SECGCurveNames}}
}
```

The object identifier id-ecPublicKey designates an elliptic curve public key. It is defined by the following (after ANSI X9.62 [ANS05b][ANS98b, 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 following information object of class ALGORITHM indicates the type of the parmaters field of an AlgorithmIdentifier {} containing the OID id-ecPublicKeyRestricted.

```
ecPublicKeyTypeRestricted ALGORITHM ::= {
     OID id-ecPublicKeyTypeRestricted PARMS ECPKRestrictions
}
```

The OID id-ecPublicKeyTypeRestricted is used to identify a public key that has restrictions on which ECC algorithms it can be used with.

```
id-ecPublicKeyTypeRestricted OBJECT IDENTIFIER ::= {
id-publicKeyType restricted(2) }
```

The type ECPKRestrictions identifies the restrictions on the algorithms that can be used with a

given elliptic curve public key.

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

The type `ECCAlgorithms` is used to identify one or more ECC algorithms, possibly, but not necessarily, in an order of preference.

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

The type `ECCAlgorithm` is a constrained instance of the parameterized type `AlgorithmIdentifier {}`, and is used to identify an ECC algorithm.

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

The component `ECDomainParameters` 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`.

The following information object of class `ALGORITHM` indicates the type of the parmaters field of an `AlgorithmIdentifier {}` containing the OID `id-ecPublicKeySupplemented`.

```
ecPublicKeyTypeSupplemented ALGORITHM ::= {
     OID id-ecPublicKeyTypeSupplemented PARMS ECPKSupplements
}
```

The OID `id-ecPublicKeyTypeSupplemented` is used to identify a public key that has restrictions on which ECC algorithms it can be used with.

```
id-ecPublicKeyTypeSupplemented OBJECT IDENTIFIER ::= { iso(1)
identified-organization(3) certicom(132) schemes(1) supplementalPoints(0) }
```

The type `ECPKSupplements` identifies the supplements (and restrictions) on the algorithms that can be used with a given elliptic curve public key.

```
ECPKSupplements ::= SEQUENCE {
     ecDomain ECDomainParameters {{ SECGCurveNames }},
```

```
    eccAlgorithms ECCAlgorithms,
    eccSupplements ECCSupplements }
```

The type ECCSupplements serves to provide a list of multiples of the public key. These multiples can be used to accelerate the public key operations necessary with that public key.

```
ECCSupplements ::= CHOICE {
    namedMultiples [0] NamedMultiples,
    specifiedMultiples [1] SpecifiedMultiples
}
NamedMultiples ::= SEQUENCE {
    multiples OBJECT IDENTIFIER,
    points SEQUENCE OF ECPoint }
SpecifiedMultiples ::= SEQUENCE OF SEQUENCE {
    multiple INTEGER,
    point ECPoint }
```

## C.4 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 ::= SEQUENCE {
    version INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
    privateKey OCTET STRING,
    parameters [0] ECDomainParameters {{ SECGCurveNames }} 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.

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 [RSA93]. In such a case, the value of the component `privateKeyAlgorithm` within `PrivateKeyInfo` shall be `id-ecPublicKey` as discussed in section C.3 above.

## C.5 Signature and Key Establishment Schemes

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 [BJP99]), 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 either contain `NULL` or be absent. The parameters component should be omitted. (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 sha1(1)}
ecdsa-with-Recommended OBJECT IDENTIFIER ::= { id-ecSigType recommended(2) }
ecdsa-with-Specified OBJECT IDENTIFIER ::= { id-ecSigType specified(3)}
ecdsa-with-Sha224 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 1 }
ecdsa-with-Sha256 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 2 }
ecdsa-with-Sha384 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 3 }
ecdsa-with-Sha512 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 4 }
ecdsa-SHA1 ALGORITHM ::= { OID ecdsa-with-SHA1 PARMS NULL }
id-ecSigType OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
```

The information object set `ECDSAAlgorithmSet` specifies how the object identifiers above are to be used in algorithm identifiers and also serves to constrain the set of algorithms specifiable in this ASN.1 syntax, when using ECDSA.

```
ECDSAAlgorithmSet ALGORITHM ::= {
    {OID ecdsa-with-SHA1} |
    {OID ecdsa-with-SHA1 PARMS NULL} |
    {OID ecdsa-with-Recommended} |
    {OID ecdsa-with-Recommended PARMS NULL} |
    {OID ecdsa-with-Specified PARMS HashAlgorithm } |
    {OID ecdsa-with-Sha224} |
    {OID ecdsa-with-Sha256} |
```

```
    {OID ecdsa-with-Sha384} |
    {OID ecdsa-with-Sha512} ,
    ...  -- More algorithms need to be added
}
```

The information object set ECCAlgorithmSet specifies the ECC algorithms that can be identied with this syntax.

```
ECCAlgorithmSet ALGORITHM ::= {
    ECDSAAlgorithmSet |
    ECDHAlgorithmSet |
    ECMQVAlgorithmSet |
    ECIESAlgorithmSet |
    ECWKTAlgorithmSet ,
    ...
}
```

The information object set ECDHAlgorithmSet used above is defined below.

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

The information object set ECMQVHAlgorithmSet used above is defined below.

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

The object identifiers used in the two information object sets above are given below.

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

```
identified-organization(3) certicom(132) schemes(1) }
dhSinglePass-stdDH-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 2}
dhSinglePass-cofactorDH-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 3}
mqvSinglePass-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 16}
mqvFull-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 17}
dhSinglePass-cofactorDH-recommendedKDF OBJECT IDENTIFIER ::=
{secg-scheme 1}
dhSinglePass-cofactorDH-specifiedKDF OBJECT IDENTIFIER ::=
{secg-scheme 2}
mqvSinglePass-recommendedKDF OBJECT IDENTIFIER ::= {secg-scheme 3}
mqvSinglePass-specifiedKDF OBJECT IDENTIFIER ::= {secg-scheme 4}
mqvFull-recommendedKDF OBJECT IDENTIFIER ::= {secg-scheme 5}
mqvFull-specifiedKDF OBJECT IDENTIFIER ::= {secg-scheme 6}
```

The object identifiers above that end in `recommendedKDF` indicated that key derivation to use is the default for the associated elliptic curve domain parameters.

The type `KeyDerivationFunction` is given below.

```
KeyDerivationFunction ::= HashAlgorithm
```

The information object set `ECEISAlgorithmSet` specifies how one identifies ECIES.

```
ECIESAlgorithmSet ALGORITHM ::= {
     {OID ecies-recommendedParameters} |
     {OID ecies-specifiedParameters PARMS ECIESParameters} ,
     ...  -- Future combinations may be added
}
```

The object identifiers given above are:

```
ecies-recommendedParameters OBJECT IDENTIFIER ::= {secg-scheme 7}
ecies-specifiedParameters OBJECT IDENTIFIER ::= {secg-scheme 8}
```

The type `ECIESParameters` is defined below.

```
ECIESParameters ::= KeyDerivationFunction
```

The information object set `ECWKTAlgorithmSet` specifies how one identifies elliptic curve wrapped key transport, if one is using the scheme as a single unit, not as a combination of the key. Typically, one may identify a wrapped key transport scheme separately as a combination of a key agreement schemes and key wrap scheme.

```
ECWKTAlgorithmSet ALGORITHM ::= {
     {OID ecwkt-recommendedParameters} |
     {OID ecwkt-specifiedParameters PARMS ECWKTParameters} ,
     ...  -- Future combinations may be added
```

}

The object identifiers given above are:

```
ecwkt-recommendedParameters OBJECT IDENTIFIER ::= {secg-scheme 9}
ecwkt-specifiedParameters OBJECT IDENTIFIER ::= {secg-scheme 10}
```

The type ECWKTParameters are defined below.

```
ECWKTParameters ::= KeyDerivationFunction
```

The actual value of an ECDSA signature computed using SHA-1, that is, a signature identified by ecdsa-with-SHA1 or any other of the above identifiers, 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.

The actual value of an ECIES ciphertext may be encoded in ASN.1 with the following type.

```
ECIES-Ciphertext-Value ::= SEQUENCE {
     ephemeralPublicKey ECPoint,
     symmetricCiphertext OCTET STRING,
     macTag OCTET STRING
}
```

## C.6 Module

To be revised.

The following comprimes the ASN.1 module for all the items specified in this standard, including those that may have been defined in other modules.

```
SEC1-v1-5 {
     iso(1) identified-organization(3) certicom(132) module(1) ver(2)
}
DEFINITIONS EXPLICIT TAGS ::= BEGIN
     --
     -- EXPORTS ALL;
     --
FieldID { FIELD-ID:IOSet } ::= SEQUENCE { -- Finite field
     fieldType FIELD-ID.&id({IOSet}),
```

```
    parameters FIELD-ID.&Type({IOSet}{@fieldType})
}
FIELD-ID ::= TYPE-IDENTIFIER
FieldTypes FIELD-ID ::= {
    { Prime-p IDENTIFIED BY prime-field } |
    { Characteristic-two IDENTIFIED BY characteristic-two-field }
}
prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }
Prime-p ::= INTEGER -- Field of size p.
id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1)}
ansi-X9-62 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) 10045
}
characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }
Characteristic-two ::= SEQUENCE {
    m INTEGER, -- Field size 2^m
    basis CHARACTERISTIC-TWO.&id({BasisTypes}),
    parameters CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}
CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
BasisTypes CHARACTERISTIC-TWO::= {
    { NULL IDENTIFIED BY gnBasis } |
    { Trinomial IDENTIFIED BY tpBasis } |
    { Pentanomial IDENTIFIED BY ppBasis },
    ...
  }
gnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
id-characteristic-two-basis OBJECT IDENTIFIER ::= {
    characteristic-two-field basisType(3)
}
Trinomial ::= INTEGER
Pentanomial ::= SEQUENCE {
    k1 INTEGER, -- k1 > 0
    k2 INTEGER, -- k2 > k1
    k3 INTEGER -- k3 > k2
}
FieldElement ::= OCTET STRING
ECDomainParameters{ECDOMAINCURVES:IOSet} ::= CHOICE {
    specifiedecParameters SpecifiedECDomainECParameters,
    namedCurve CURVESECDOMAIN.&id({IOSet}),
    implicitCA NULL
}
SpecifiedECDomainECParameters ::= SEQUENCE {
```

```
     version SpecifiedECDomainVersionINTEGER { ecpVer1(1) } (ecdpVer1 | ecdpVer2 |
ecdpVer3, ...),
     fieldID FieldID {{FieldTypes}},
     curve Curve,
     base ECPoint,
     order INTEGER,
     cofactor INTEGER OPTIONAL,
     hash HashAlgorithm OPTIONAL,
      ...
}
SpecifiedECDomainVersion ::= INTEGER {
     ecdpVer1(1),
     ecdpVer2(2),
     ecdpVer3(3)
}
 Curve ::= SEQUENCE {
     a FieldElement,
     b FieldElement,
     seed BIT STRING OPTIONAL
     -- Shall be present if used in SpecifiedECDomain with version equal to ecdpVer2
or ecdpVer3
}
ECPoint ::= OCTET STRING
CURVESECDOMAIN ::= CLASS {
     &id OBJECT IDENTIFIER UNIQUE
}
WITH SYNTAX { ID &id }
SECGCurveNames ECDOMAINCURVES::= {
     ...  -- named curves
}
HashAlgorithm ::= AlgorithmIdentifier {{ HashFunctions }}
 HashFunctions ALGORITHM ::= {
     {OID sha-1} | {OID sha-1 PARMS NULL } |
     {OID id-sha224} | {OID id-sha224 PARMS NULL } |
     {OID id-sha256} | {OID id-sha256 PARMS NULL } |
     {OID id-sha384} | {OID id-sha384 PARMS NULL } |
     {OID id-sha512} | {OID id-sha512 PARMS NULL } ,
     ...  -- Additional hash functions may be added in the future }
 sha-1 OBJECT IDENTIFIER ::= {iso(1) identified-organization(3)
oiw(14) secsig(3) algorithm(2) 26}
id-sha OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16) us(840)
organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) }
id-sha224 OBJECT IDENTIFIER ::= { id-sha 4 }
id-sha256 OBJECT IDENTIFIER ::= { id-sha 1 }
id-sha384 OBJECT IDENTIFIER ::= { id-sha 2 }
```

```
id-sha512 OBJECT IDENTIFIER ::= { id-sha 3 }
 SubjectPublicKeyInfo ::= SEQUENCE {
     algorithm AlgorithmIdentifier {{ECPKAlgorithms}},
     subjectPublicKey BIT STRING
}
AlgorithmIdentifier{ ALGORITHM:IOSet } ::= SEQUENCE {
     algorithm ALGORITHM.&id({IOSet}),
     parameters ALGORITHM.&Type({IOSet}{@algorithm})
}
ALGORITHM ::= CLASS {
     &id OBJECT IDENTIFIER UNIQUE,
     &Type OPTIONAL
}
     WITH SYNTAX { OID &id [PARMS &Type] }
ECPKAlgorithms ALGORITHM ::= {
     ecPublicKeyType |
     ecPublicKeyTypeRestricted |
     ecPublicKeyTypeSupplemented ,
     ...
}
ecPublicKeyType ALGORITHM ::= {
     OID id-ecPublicKey PARMS ECDomainParameters {{SECGCurveNames}}
}
id-ecPublicKey OBJECT IDENTIFIER ::= { id-publicKeyType 1 }
id-publicKeyType OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) }
ecPublicKeyTypeRestricted ALGORITHM ::= {
     OID id-ecPublicKeyTypeRestricted PARMS ECPKRestrictions
}
 id-ecPublicKeyTypeRestricted OBJECT IDENTIFIER ::= {
id-publicKeyType restricted(2) }
 ECPKRestrictions ::= SEQUENCE {
     ecDomain ECDomainParameters {{ SECGCurveNames }},
     eccAlgorithms ECCAlgorithms
}
 ECCAlgorithms ::= SEQUENCE OF ECCAlgorithm
 ECCAlgorithm ::= AlgorithmIdentifier {{ECCAlgorithmSet}}
 ecPublicKeyTypeSupplemented ALGORITHM ::= {
     OID id-ecPublicKeyTypeSupplemented PARMS ECPKSupplements
}
 id-ecPublicKeyTypeSupplemented OBJECT IDENTIFIER ::= { iso(1)
identified-organization(3) certicom(132) schemes(1) supplementalPoints(0) }
 ECPKSupplements ::= SEQUENCE {
     ecDomain ECDomainParameters {{ SECGCurveNames }},
     eccAlgorithms ECCAlgorithms,
     eccSupplements ECCSupplements }
```

```
 ECCSupplements ::= CHOICE {
     namedMultiples [0] NamedMultiples,
     specifiedMultiples [1] SpecifiedMultiples
}
NamedMultiples ::= SEQUENCE {
     multiples OBJECT IDENTIFIER,
     points SEQUENCE OF ECPoint }
SpecifiedMultiples ::= SEQUENCE OF SEQUENCE {
     multiple INTEGER,
     point ECPoint }
 ECPrivateKey ::= SEQUENCE {
     version INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
     privateKey OCTET STRING,
     parameters [0] ECDomainParameters {{ SECGCurveNames }} OPTIONAL,
     publicKey [1] BIT STRING OPTIONAL
}
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType sha1(1)}
ecdsa-with-Recommended OBJECT IDENTIFIER ::= { id-ecSigType recommended(2) }
ecdsa-with-Specified OBJECT IDENTIFIER ::= { id-ecSigType specified(3)}
ecdsa-with-Sha224 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 1 }
ecdsa-with-Sha256 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 2 }
ecdsa-with-Sha384 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 3 }
ecdsa-with-Sha512 OBJECT IDENTIFIER ::= { id-ecSigType specified(3) 4 }
ecdsa-SHA1 ALGORITHM ::= { OID ecdsa-with-SHA1 PARMS NULL }
id-ecSigType OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
ECDSAAlgorithmSet ALGORITHM ::= {
     {OID ecdsa-with-SHA1} |
     {OID ecdsa-with-SHA1 PARMS NULL} |
     {OID ecdsa-with-Recommended} |
     {OID ecdsa-with-Recommended PARMS NULL} |
     {OID ecdsa-with-Specified PARMS HashAlgorithm } |
     {OID ecdsa-with-Sha224} |
     {OID ecdsa-with-Sha256} |
     {OID ecdsa-with-Sha384} |
     {OID ecdsa-with-Sha512} ,
     ...  -- More algorithms need to be added
}
 ECCAlgorithmSet ALGORITHM ::= {
     ECDSAAlgorithmSet |
     ECDHAlgorithmSet |
     ECMQVAlgorithmSet |
     ECIESAlgorithmSet |
     ECWKTAlgorithmSet ,
     ...
}
```

```
 ECDHAlgorithmSet ALGORITHM ::= {
     {OID dhSinglePass-stdDH-sha1kdf} |
     {OID dhSinglePass-stdDH-sha1kdf PARMS NULL} |
     {OID dhSinglePass-cofactorDH-sha1kdf} |
     {OID dhSinglePass-cofactorDH-sha1kdf PARMS NULL} |
     {OID dhSinglePass-cofactorDH-recommendedKDF} |
     {OID dhSinglePass-cofactorDH-specifiedKDF PARMS KeyDerivationFunction} ,
     ...  -- Future combinations may be added
}
 ECMQVAlgorithmSet ALGORITHM ::= {
     {OID mqvSinglePass-sha1kdf} |
     {OID mqvSinglePass-recommendedKDF} |
     {OID mqvSinglePass-specifiedKDF PARMS KeyDerivationFunction} |
     {OID mqvFull-sha1kdf} |
     {OID mqvFull-recommendedKDF} |
     {OID mqvFull-specifiedKDF PARMS KeyDerivationFunction} ,
     ...  -- Future combinations may be added
}
 x9-63-scheme OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) ansi-x9-63(63) schemes(0) }
secg-scheme OBJECT IDENTIFIER ::= { iso(1)
identified-organization(3) certicom(132) schemes(1) }
dhSinglePass-stdDH-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 2}
dhSinglePass-cofactorDH-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 3}
mqvSinglePass-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 16}
mqvFull-sha1kdf OBJECT IDENTIFIER ::= {x9-63-scheme 17}
dhSinglePass-cofactorDH-recommendedKDF OBJECT IDENTIFIER ::=
{secg-scheme 1}
dhSinglePass-cofactorDH-specifiedKDF OBJECT IDENTIFIER ::=
{secg-scheme 2}
mqvSinglePass-recommendedKDF OBJECT IDENTIFIER ::= {secg-scheme 3}
mqvSinglePass-specifiedKDF OBJECT IDENTIFIER ::= {secg-scheme 4}
mqvFull-recommendedKDF OBJECT IDENTIFIER ::= {secg-scheme 5}
mqvFull-specifiedKDF OBJECT IDENTIFIER ::= {secg-scheme 6}
 KeyDerivationFunction ::= HashAlgorithm
 ECIESAlgorithmSet ALGORITHM ::= {
     {OID ecies-recommendedParameters} |
     {OID ecies-specifiedParameters PARMS ECIESParameters} ,
     ...  -- Future combinations may be added
}
 ecies-recommendedParameters OBJECT IDENTIFIER ::= {secg-scheme 7}
ecies-specifiedParameters OBJECT IDENTIFIER ::= {secg-scheme 8}
 ECIESParameters ::= KeyDerivationFunction
 ECWKTAlgorithmSet ALGORITHM ::= {
     {OID ecwkt-recommendedParameters} |
```

```
    {OID ecwkt-specifiedParameters PARMS ECWKTParameters} ,
    ...  -- Future combinations may be added
}
 ecwkt-recommendedParameters OBJECT IDENTIFIER ::= {secg-scheme 9}
ecwkt-specifiedParameters OBJECT IDENTIFIER ::= {secg-scheme 10}
 ECWKTParameters ::= KeyDerivationFunction
 ECDSA-Sig-Value ::= SEQUENCE {
    r INTEGER,
    s INTEGER
}
ECIES-Ciphertext-Value ::= SEQUENCE {
    ephemeralPublicKey ECPoint,
    symmetricCiphertext OCTET STRING,
    macTag OCTET STRING
}
 END
```

The following comprises the ASN.1 module for those items not defined in ANSI X9.62 [ANS98b, Section 6.1].

```
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

[ABM+03] A. Antipa, D. Brown, A. Menezes, R. Struik, and S. Vanstone, *Validation of elliptic curve public keys*, Public Key Cryptography — PKC 2003 (Y. G. Desmedt, ed.), LNCS 2567, IACR, Springer, 2003, pp. 211–223.

[ABMV93] G. Agnew, T. Beth, R. Mullin, and S. Vanstone, *Arithmetic operations in $GF(2^m)$*, J. of Cryptology **6** (1993), 3–13.

[ABR98] M. Abdullah, M. Bellare, and P. Rogaway, *DHAES: An encryption scheme based on the Diffie-Hellman problem*, 1998, Full version of [BR97] at `www-cse.ucsd.edu/users/mihir`.

[AMOV91] G. Agnew, R. Mullin, I. Onyszchuk, and S. Vanstone, *An implementation for a fast public-key cryptosystem*, J. of Cryptology **3** (1991), 63–79.

[AMV93] 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** (1993), 804–813.

[ANS98a] *ANS X9.52-1998: Triple data encryption: Modes of operation*, 1998, `webstore.ansi.org/ansidocstore/default.asp`.

[ANS98b] *ANS X9.62-1998: Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)*, 1998, Revision scheduled for 2005. `webstore.ansi.org/ansidocstore/default.asp`.

[ANS01a] *ANS X9.63-2001: Public-key cryptography for the financial services industry: Key agreement and key transport using elliptic curve cryptography*, 2001, `webstore.ansi.org/ansidocstore/default.asp`.

[ANS01b] *ANS X9.71-2001: Key hash message authentication code*, 2001, Not currently available.

[ANS03] *Draft ANS X9.102-2003: Symmetric key cryptography for the financial services industry: Part 1: Wrapping of keys and associated data*, 2003, Draft.

[ANS05a] *Draft ANS X9.82: Random number generation*, 2005, Tentative organization: Part 1: Overview; Part 2: Entropy Sources; Part 3: Deterministic Algorithms; Part 4: Complete Systems.

[ANS05b] *Draft ANS X9.62-2005: Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)*, 2005, Draft revision.

[BCK98] 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.

[BDL97] D. Boneh, R. A. DeMillo, and R. J. Lipton, *On the importance of checking cryptographic protocols for faults*, Advances in Cryptology – EUROCRYPT '97 (W. Fumy, ed.), LNCS 1233, IACR, Springer, 1997, pp. 37–51.

[BDR+96] 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.

[BHP02] L. Bassham, R. Housley, and W. Polk, *RFC 3279: Algorithms and identifiers for the internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile*, IETF, April 2002, `www.ietf/rfc/rfc3279.txt`.

[BJP99] L. Bassham, D. Johnson, and W. Polk, *Representation of elliptic curve digital signature algorithm (ECDSA) keys and signatures in internet X.509 public key infrastructure certificates*, IETF, June 1999, Expired Internet-Draft.

[BL96] D. Boneh and R. J. Lipton., *Algorithms for black-box fields and their application to cryptography*, In Koblitz [Kob96], pp. 283–297.

[Bon98] D. Boneh, *The decision Diffie-Hellman problem*, Algorithmic Number Theory III (J. P. Buehler, ed.), LNCS 1423, Springer, 1998, pp. 48–63.

[BR97] M. Bellare and P. Rogaway, *Minimizing the use of random oracles in authenticated encryption schemes*, Proceedings of PKS '97, 1997.

[Bro04] D. R. L. Brown, *Additional algorithms and identifiers for use of ECC with PKIX*, IETF, July 2004, `www.ietf.org/internet-drafts/draft-ietf-pkix-ecc-pkalgs-00.txt`.

[BSS99] I. Blake, G. Seroussi, and N. Smart, *Elliptic curves in cryptography*, Cambridge University Press, 1999.

[BV96] D. Boneh and R. Venkatesan, *Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes*, In Koblitz [Kob96], pp. 129–142.

[BWBL02] S. Blake-Wilson, D. Brown, and P. Lambert, *RFC 3278: Use of ECC algorithms in CMS*, IETF, April 2002, `www.ietf.org/rfc/rfc3278.txt`.

[BWJM97] S. Blake-Wilson, D. Johnson, and A. J. Menezes, *Key agreement protocols and their security analysis*, Proceedings of the 6th IMA International Conference on Cryptography and Coding, 1997, pp. 30–45.

[BWM99] S. Blake-Wilson and A. J. Menezes, *Unknown key-share attacks on the station-to-station (STS) protocol*, Public Key Cryptography — PKC '99 (H. Ideki and Y. Zheng, eds.), LNCS 1560, IACR, Springer, 1999, pp. 154–170.

[CH98] 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.

[CS98] R. Cramer and V. Shoup, *A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack*, Advances in Cryptology — CRYPTO '98 (H. Krawczyk, ed.), LNCS 1462, IACR, Springer, 1998, pp. 13–25.

[DA98] T. Dierks and B. Anderson, *ECC cipher suites for TLS*, IETF, March 1998, Expired Internet Draft.

[DA99] T. Dierks and C. Allen, *The TLS protocol, version 1.0*, IETF, January 1999, `www.ietf.org/rfc/rfc2246.txt`.

[DH76] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **IT-22** (1976), no. 6, 644–654.

[DvOW92] W. Diffie, P.C. van Oorschot, and M.J. Wiener, *Authentication and authenticated key exchanges*, Designs, Codes and Cryptography **2** (1992), 107–125.

[Dwo04] Morris Dworkin, *Request for review of key wrap algorithms*, ASC X9, November 2004, `eprint.iacr.org/2004/340`.

[ECC] *ECC Challenge*, Details at `www.certicom.com`.

[ElG85] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **IT-31** (1985), 469–472.

[Fin99] Financial Services Technology Consortium, *Financial services markup language*, August 1999, Working Draft.

[FIP93a] NIST, *FIPS 186: Digital signature standard*, 1993, `csrc.nist.gov/publications/fips`.

[FIP93b] NIST, *FIPS 46-2: Data encryption standard*, 1993, `csrc.nist.gov/publications/fips`.

[FIP95] NIST, *FIPS 180-1: Secure hash standard*, 1995, `csrc.nist.gov/publications/fips`.

[FIP01a] NIST, *FIPS 186-2: Digital signature standard (change notice)*, October 2001, `csrc.nist.gov/publications/fips`.

[FIP01b] NIST, *FIPS 197: Advanced encryption standard (change notice)*, October 2001, `csrc.nist.gov/publications/fips`.

[FIP04] NIST, *FIPS 180-2: Secure hash standard (change notice)*, February 2004, `csrc.nist.gov/publications/fips`.

[FR94] G. Frey and H.-G. Rüuck, *A remark concerning m-divisibility and the discrete logarithm problem in the divisor class group of curves*, Mathematics of Computation **62** (1994), 865–874.

[GBWM+04] V. Gupta, S. Blake-Wilson, B. Möller, C. Hawk, and N. Bolyard, *Internet-draft: ECC ciphersuites for TLS*, IETF, December 2004, `www.ietf.org/internet-drafts/draft-ietf-tls-ecc-07.txt`.

[GLV] R. Gallant, R. Lambert, and S. A. Vanstone, *Improving the parallelized Pollard lambda search on binary anomalous curves*, Mathematics of Computation, To appear.

[GS] S. D. Galbraith and N. P. Smart, *A cryptographic application of the Weil descent*, To appear.

[HC98] D. Harkins and D. Carrel, *RFC 2409: The internet key exchange*, IETF, 1998, `www.ietf.org/rfc/rfc2409.txt`.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo, *RFC 2459: Internet X.509 public key infrastructure certificate and CRL profile*, IETF, 1999, `www.ietf.org/rfc/rfc2459.txt`.

[Hou99] R. Housley, *RFC 2630: Cryptographic message syntax*, IETF, June 1999, `www.ietf.org/rfc/rfc2630.txt`.

[IEE97] *IEEE P1363a: Standard for public-key cryptography - addendum*, July 1997, Working Document.

[IEE99] *IEEE P1363: Standard for public-key cryptography*, July 11 1999, Working Draft.

[IEE00] *IEEE Std 1363-2000: Standard specifications for public-key cryptography*, 2000, `standards.ieee.org/catalog/olis/busarch.html`.

[IEE04] *IEEE Std 1363A-2004: Standard specifications for public-key cryptography — amendment 1: Additional techniques*, 2004, `standards.ieee.org/catalog/olis/busarch.html`.

[ISO] *ISO/IEC 14888-3: Information technology — security techniques — digital signatures with appendix — part 3: Certificate-based mechanisms.*

[ISO98a] *ISO/IEC 15946-1: Information technology — security techniques — cryptographic techniques based on elliptic curves — part 1: General*, 1998, Working draft.

[ISO98b] *ISO/IEC 15946-2: Information technology — security techniques — cryptographic techniques based on elliptic curves — part 2: Digital signatures*, 1998, Working draft.

[ISO98c] *ISO/IEC 15946-3: Information technology — security techniques — cryptographic techniques based on elliptic curves — part 3: Key establishment*, 1998, Working draft.

[ITUa] *ITU-T recommendation X.681: Information technology — abstract syntax notation one (ASN.1): Information object specification*, Equivalent to ISO/IEC 8824-2.

[ITUb] *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.

[Joh96] D. Johnson, *Diffie-Hellman key agreement small subgroup attack, a contribution to X9F1 by certicom*, July 16 1996.

[Jun93] D. Jungnickel, *Finite fields: Structure and arithmetics*, B. I. Wissenschaftsverlag, Mannheim, 1993.

[Kal98] B. Kaliski, *MQV vulnerability*, Posting to ANSI X9F1 and IEEE P1363 newsgroups, 1998.

[Kau05] C. Kaufman, *Internet key exchange (IKEv2) protocol*, IETF, March 2005, `www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-17.txt`.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti, *RFC 2104: HMAC: Keyed hashing for message authentication*, IETF, 1997, `www.ietf.org`.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun, *Differential power analysis*, Advances in Cryptology — CRYPTO '99 (M. Wiener, ed.), LNCS 1666, IACR, Springer, 1999, pp. 388–397.

[KMV] N. Koblitz, A.J. Menezes, and S.A. Vanstone, *The state of elliptic curve cryptography*, Designs, Codes and Cryptography, To appear.

[Knu81] D. Knuth, *The art of computer programming — seminumerical algorithms*, second ed., vol. 2, Addison-Wesley, 1981.

[Kob87] N. Koblitz, *Elliptic curve cryptosystems*, vol. 48, 1987.

[Kob92] N. Koblitz, *CM-curves with good cryptographic properties*, Advances in Cryptology — CRYPTO '91 (J. Feigenbaum, ed.), LNCS 576, IACR, Springer, 1992, pp. 279–287.

[Kob94] N. Koblitz, *A course in number theory and cryptography*, second ed., Springer, 1994.

[Kob96] N. Koblitz (ed.), *Advances in cryptology — CRYPTO '96*, LNCS 1109, IACR, Springer, 1996.

[Koc96] P. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, In Koblitz [Kob96], pp. 104–113.

[LL97] C. H. Lim and P.J. Lee, *A key recovery attack on discrete log-based schemes using a prime order subgroup*, Advances in Cryptology — CRYPTO '97 (B. S. Kaliski, ed.), LNCS 1294, IACR, Springer, 1997, pp. 249–263.

[LMQ+98] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, *An efficient protocol for authenticated key agreement*, Research Report 98-05, Department of Combinatorics & Optimization, University of Waterloo, March 1998, `www.cacr.math.uwaterloo.ca`.

[LN87] R. Lidl and H. Neiderreiter, *Finite fields*, Cambridge University Press, 1987.

[LZ94] G. Lay and H. Zimmer, *Constructing elliptic curves with given group order over large finite fields*, Algorithmic Number Theory (1994), 250–263.

[McE87] R. J. McEliece, *Finite fields for computer scientists and engineers*, Kluwer Academic Publishers, 1987.

[Men93] A. J. Menezes, *Elliptic curve public key cryptosystems*, Kluwer Academic Publishers, 1993.

[Mil92] V. Miller, *Uses of elliptic curves in cryptography*, Advances in Cryptology — CRYPTO '85 (H. C.. Williams, ed.), LNCS 218, IACR, Springer, 1992, pp. 417–426.

[MOV93] 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** (1993), 1639–1646.

[MQV95] A.J. Menezes, M. Qu, and S.A. Vanstone, *Some new key agreement protocols providing implicit authentication*, 2nd Workshop on Selected Areas in Cryptography - SAC '95, May 1995.

[MvOV97] A. J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.

[NIS99] NIST, *Recommended elliptic curves for federal government use*, July 1999, `csrc.nist.gov/encryption`.

[NIS01a] NIST, *AES key wrap specification*, November 2001, `csrc.nist.gov/CryptoToolkit/kms/AES_key_wrap.pdf`.

[NIS01b] NIST, *SP 800-38A: Recommendation for block cipher modes of operation*, December 2001, `csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`.

[NIS03] NIST, *SP 800-56: Recommendation on key establishment schemes*, January 2003, Draft 2.0. `csrc.nist.gov/CryptoToolkit/kms/keyschemes-Jan03.pdf`.

[NR93] K. Nyberg and R. Rueppel, *A new signature scheme based on DSA giving message recovery*, 1st ACM Conference on Computer and Communications Security, ACM Press, 1993, pp. 58–61.

[NR96] K. Nyberg and R. Rueppel, *Message recovery for signature schemes based on the discrete logarithm problem*, Designs, Codes and Cryptography **7** (1996), 61–81.

[Odl95] A. Odlyzko, *The future of integer factorization*, CryptoBytes **1** (1995), no. 2, 5–12.

[PH78] 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** (1978), 106–110.

[Pol78] J. Pollard, *Monte Carlo methods for index computation mod p*, Mathematics of Computation **32** (1978), 918–924.

[PS96] D. Pointcheval and J. Stern, *Security proofs for signatures*, Advances in Cryptology — EUROCRYPT '96 (U. Maurer, ed.), LNCS 1070, IACR, Springer, 1996, pp. 387–398.

[RSA93] RSA Laboratories, *PKCS #8: Private-key information syntax standard*, November 1993, `www.rsa.com/rsalabs/pubs/PKCS`.

[SA] T. Satoh and K. Araki, *Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves*, To appear.

[Sch85] R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*, Mathematics of Computation **44** (1985), 483–494.

[Sch91] C. P. Schnorr, *Efficient signature generation by smart cards*, J. of Cryptology **4** (1991), 161–174.

[SEC99a] SECG, *GEC 1: Recommended elliptic curve domain parameters*, September 1999, Working Draft. `www.secg.org`.

[SEC99b] SECG, *GEC 2: Test vectors for SEC 1*, September 1999, Working Draft. `www.secg.org`.

[SEC00] *SEC 2: Recommended elliptic curve domain parameters*, September 2000, Version 1.0. `www.secg.org`.

[Sem] I. Semaev, *Evaluation of discrete logarithm on some elliptic curves*, Mathematics of Computation, To appear.

[SH02] J. Schaad and R. Housley, *RFC 3394: Advanced encryption standard (AES) key wrap algorithm*, IETF, September 2002, `www.ietf.org/rfc/rfc3394.txt`.

[Sil85] J. Silverman, *The arithmetic of elliptic curves*, Springer, 1985.

[Sma] N. Smart, *The discrete logarithm problem on elliptic curves of trace one*, J. of Cryptology, To appear.

[Vau96] S. Vaudenay, *Hidden collisions on DSS*, In Koblitz [Kob96], pp. 83–88.

[vOW94] 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, ACM Press, 1994, pp. 210–218.

[WAP99] *WAP WTLS: Wireless application protocol wireless transport layer security specification*, February 1999.

[WZ99] M. Wiener and R.J. Zuccherato, *Fast attacks on elliptic curve cryptosystems*, 5th Annual Workshop on Selected Areas in Cryptography: SAC '98 (S. Tavares and H. Meijer, eds.), LNCS 1556, Springer, 1999, pp. 190–200.